# Controlling Molecules with Lasers and Lasers with Molecules

by

Jason Matthew Taylor

B.E., Vanderbilt University (1999)
S.M., Massachusetts Institute of Technology (2002)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[February 2007]
September 2006





Author .....
Program in Media Arts and Sciences,
School of Architecture and Planning
September 25, 2006



Certified by .....
Neil Gershenfeld
Associate Professor
Thesis Supervisor

Accepted by .....
Andrew B. Lippman
Chairperson, Departmental Committee on Graduate Students

# Controlling Molecules with Lasers and Lasers with Molecules

by

Jason Matthew Taylor

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on September 25, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

## Abstract

I investigate quantum control of spin in molecules using shaped ultrafast lasers and the dynamics of those lasers when their cavities are modified to include programmable molecular masks. The ability to control quantum phenomena has had several large successes over the last decade. This field, known as Quantum Control, uses closed loop learning algorithms to shape ultrashort laser pulses in order to produce a desired state or state change. Interesting pulse shapes have been able to break chemical bonds, drive chemical reactions, selectively excite molecular states, and most recently, control photoisomerization in proteins [1, 2, 3]. In this thesis I began by seeking to apply this technique to manipulate spin. In our early work we pursued polarizing electron spins and nuclear spins for NMR Quantum Computation. We studied the electron spin triplet state properties of several molecules. Through this work we recognized that the laser and pulse shaper we were using could be modified to utilize the triplet properties of our molecules. We created a molecular triplet state spatial light modulator (SLM) to be used both outside and inside the laser cavity for ultrafast pulse shaping. The SLM consists of a liquid or thin film sample with a strong triplet state absorption. The molecule is selected to be transparent to the target light before pumping and strongly absorptive when pumped into the triplet state. The sample is exposed to laser light reflected off of a DMD chip to produce a 2D pattern to spatially populate the triplet ground state. This is, to our knowledge, the first triplet state ultrafast pulse shaper and the first all-optical inter-cavity spatial frequency modulator.

Thesis Supervisor: Neil Gershenfeld
Title: Associate Professor

# Acknowledgments

Throughout this work I have benefited from the guidance of my graduate advisor at MIT, Neil Gershenfeld and my co-advisor at the University of Michigan, Phil Bucksbaum. I would like to thank Phil Bucksbaum for the use of his ultrafast lasers and for his support over the last several years. I would also like to thank my thesis reader, Erich Ippen for his insight into laser cavity dynamics.

I would like to thank Phil's group in Michigan: Brett Pearson, Andrei Florean, John Caraher, Catherine Herne, Daniel Morris, Joel Murray, Emily Peterson, James White, and Alex Prociuk.

The lab is an amazing place, the group Neil assembled at the beginning of my graduate career are some of the most talented and thoughtful people I've met. I would like to thank: Yael Maguire, Ben Recht, Rehmi Post, Matt Reynolds, Ben Vigoda for their support and friendship over the years. When I returned to MIT from Michigan after two years of ultrafast work, I found that almost everyone had graduated and a new group had assembled. Over the last two years I've come to know the new group. Physics and Media is in good hands with the new generation: Manu Prakash, Ara Knaian, Amy Sun and George Popescu. Thank you.

I would like to thank Susan Murphy-Bottari without her help I wouldn't know when or where or how to do much of anything around the lab.

Two chemists put up with my endless questions: Brian Chow and Tadd Kippeny. Thank you Brian for your help and friendship over the last year or two. Tadd and I have been friends since he was a new graduate student at Vanderbilt and I was an undergraduate researcher. It is safe to say that almost everything I know about chemistry I learned from Tadd. Thank you Tadd.

For his friendship over the last decade, I would like to thank Hui-Hui. He has contributed a poem for each of my theses.

Finally, I would like to thank my family. I thank my parents for their love and support. And to my love, Marianna: Thank you for your support over the last few months while I finished this degree. You're my life.

**Controlling Molecules with Lasers and Lasers with Molecules**

by

Jason Matthew Taylor

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2006

Thesis Advisor .....

Neil Gershenfeld
Associate Professor, Media Arts and Sciences
Director Center of Bits and Atoms
MIT

Thesis Reader .....

Erich Ippen
Elihu Thomson Professor of Electrical Engineering
Professor of Physics
MIT

Thesis Reader .....

Philip Bucksbaum
Professor
Physics and Applied Physics Departments
Stanford University

## Controlling Lasers with Metaphors

Our butcher was as exact
with meat, disappearing
into a white room, holding
a bag of cut lamb shanks,
mostly solid, part liquid.
The scientist tells me
his molecules look like hemoglobin.
But only their shape
is like blood, he says,
it's just a metaphor.
He shows me his hands:
clean, no blood.

It is still January.
The experiment drags on,
like trying for a child.
Nobody's heard anything
for over a month. How will
we know the wait is over?
When everything works, he says,
the laser chirps like a bird.
He thinks for a while,
and then he chirps for us:
chirrrp, chirrrp, chirrrp.

He says: Two wars have been fought
since my research started,
making it epic in nature.
If this were literature,
the subject would be love
and I would be the protagonist.
If you don't believe me,
don't forget that,
in the middle of one experiment,
I married my lover.

Tung-Hui Hu

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

During the first two years of my doctoral studies I used lasers to populate the triplet states of molecules in a quantum control experiment. Our ultrafast system consisted of a Kerr lens mode-locked oscillator, two amplifiers, a pulse shaper and a computer running a learning algorithm to discover useful pulse shapes. It was through this work that we became interested in the information processing capacity of the ring-up of an ultrafast laser and in the ability of triplet-state molecules to modify those dynamics. Many of the principles involved in the operation of the first experiment translated directly into tools I used for the construction of the molecular triplet state pulse shapers presented here. This work represents the first steps towards using molecules to dynamically control laser cavity ring-up. Ultimately we envision using this new access into the laser cavity to remove the computer from the quantum control experiment. By feeding the residual light from an extra cavity experiment back into the laser cavity it may be able to perform an all-optical search for efficient pulse shapes.

## 1.1 The Laser

The first Kerr lens mode-locked (KLM) laser was realized in 1991 with a pulse width of 60 femtoseconds (1 fs = $10^{-15}$s) [11]. Lasers producing pulses this short are known as ultrafast lasers. At this time scale most molecular dynamics are frozen–molecules

do not spin, vibrate, or even emit light at a rate faster than these laser pulses. This precise time resolution makes ultrafast lasers a invaluable tool for studying molecular dynamics [12].

Understanding these lasers relies on descriptions in the frequency and time domains. KLM lasers 'mode-lock' when the self-focusing effect in the Kerr medium causes the losses in a cavity to decrease with increasing power in the medium. This feedback excites many modes of the cavity in phase with each other resulting in a high intensity ultrafast pulse. The ring-up of the ultrafast pulse reaches a stable point when the self-phase modulation is balanced with the self-amplitude modulation and cavity dispersion.

Typically these lasers are used in a pump-probe configuration. One ultrafast pulse can be split into two and amplified. The first pulse is used to initiate, or pump, a chemical process in a sample while the second is delayed and later used to measure, or probe, the response of the sample. The delay is created by directing the second pulse to travel a longer distance before arriving at the sample. By varying distance of the probe path by microns we attain femtosecond resolution of the timing between the two pulses. Such experiments have allowed chemists to measure fast processes such as fluorescence lifetimes and charge transfer between molecules. When trying to engineer molecular systems, information such as these rates is crucial.

## 1.2 Controlling Molecules with Lasers

A logical step after observing molecular dynamics is to attempt to control them. Femtosecond ultrafast-lasers can establish coherent states across molecules. This light can break specific bonds and drive chemical reactions. This field, known as Femtochemistry, has flourished in the last two decades. The subfield known as Quantum Control shapes ultrafast pulses in time and frequency to control quantum systems. The shape of the pulses is discovered using a learning algorithm. The algorithm–usually a genetic algorithm–directs the laser system to shine a variety of shaped pulses at a molecular sample and look for a desired behavior in the molecule. The algorithm then analyzes

the results and generates a new set of pulse shapes to shine on the molecule in the next iteration. The iterations are halted when the desired control over the molecular system is discovered. Beyond studying chemistry, Quantum Control allows us to discover how information can be manipulated, transferred, and stored in the physical states of molecules. This technique was a prime candidate for manipulating quantum computers [13].

Quantum computers implemented in Nuclear Magnetic Resonance (NMR) systems are fundamentally limited by nuclear spin polarization [14]. The complexity of the problem that can be approached is constrained by the number of qubits available. Each additional qubit for room temperature liquid NMR requires exponentially more polarization to be able to read the result of the computation. Although the cost can be made polynomial [15], polarization enhancement is required to make the prefactors tractable.

In June of 2002 we began experiments in Professor Bucksbaum's lab at the University of Michigan using a shaped ultrafast laser and an NMR with the goal of increasing nuclear spin polarization [13] for Quantum Computation. The laser system consisted of a KLM oscillator (ultrafast laser), a stretcher, pulse picker, regenerative amplifier, pulse shaper, multipass amplifier, compressor, and a computer. This system sprawled over five 8 foot by 12 foot laser tables.

We used an ultrafast optical pulse shaper to investigate the effectiveness of shaped pulses to control the creation of electron spin triplet population in a liquid. We focused on two metallo-porphyrins: Copper(II) Tetraphenylporphyrin (CuTPP) and Zinc Tetraphenylporphyrin (ZnTPP). Typical quantum efficiencies for singlet to triplet intersystem crossings in metallo-porphyrins are 67 to 88% [16]. Our goals were to selectively enhance or suppress the creation of population the triplet ground state and to selectively populate one of the three states. To find pulse shapes that accomplish this goal we employed an adaptive learning algorithm.

By understanding the KLM oscillator cavity dynamics we hope to re-create some of the functionality of this large system using only feedback into the oscillator. Part of KLM oscillator operates almost identically to the pulse shaper–the critical component

of the large system. Coincidentally, some of the molecules we studied while trying to increase nuclear spin polarization were useful in the feedback portion of our new cavities.

## 1.3 Controlling Lasers with Molecules

While working in Professor Bucksbaum's laser lab in Michigan we became interested the information processing capacity of KLM laser cavities.

> "The laser cavity finds the pulse that minimizes loss–it's like magic."
>
> Phil Bucksbaum to Neil Gershenfeld
>
> (just before my research topic changed)

The extent to which this statement is accurate indicates the usefulness of these cavities for information processing. Most optimistically Phil's statement implied that the laser can be used as an oracle to find short pulses that "solve" arbitrary loss. In Chapter 3 we will review the equations governing ultrashort pulse formation.

We found ourselves asking several questions: Could the laser cavity find the optimal pulse (least lossy) given several loss options? How complicated can the loss be before the laser doesn't work? How will the laser behave when presented with a changing loss? Could the laser, in a feedback configuration, perform gradient descent to search for a pulse shape that minimizes loss?

In Chapter 4 we will discuss laser cavity configurations to investigate each of these questions. The last question about the ability of the laser cavity to perform search was intriguing. In Chapter 5 we review the KLM cavity dynamics to try to identify a Lyapunov function that might give us some insight into what the cavity evolution minimizes–if anything.

Finally, in Chapter 6 I created a triplet state spatial light modulator to be used in an extra-cavity pulse shaper (Chapter 7) as well as an intra-cavity pulse shaper (Chapter 8).

If the loss of the cavity is linked to molecular interactions then the molecules themselves could determine the pulse shape. In the last few chapters of this thesis

we present a new all-optical spatial light modulator for ultrafast pulse shaping to be used both inside and outside of the laser cavity.

## 1.4 Background

### 1.4.1 Ultrafast Lasers

KLM lasers mode-lock when self-focusing in the Kerr medium causes the loss in a cavity to decrease with increasing pulse power. Shorter pulses experience less loss and therefore more net gain. This feedback causes mode-locking of many frequencies in phase with each other resulting in an ultrashort pulse. Using a Ti:Sapphire ($Ti:Al_2O_3$) crystal as both the gain medium and the Kerr medium as been successful and popular. The pulse reaches a steady state solution when the pulse shortening effect of the fast saturable absorber is balanced by pulse lengthing effects such as dispersion or bandwidth filtering. The soliton-like pulse that is formed is a result of a balance between GVD, SPM. Gain saturation, bandwidth filtering, and the pulse shortening effect of the Kerr media stabilize the pulse.

#### Mode-locking

The term 'Mode-locking' refers to the frequency modes in a laser cavity. Typically we think of a short laser pulse in the time domain. A Gaussian laser pulse in the time domain is also a Gaussian shape in the frequency domain. Constructing a laser cavity to excite nearby cavity modes in phase with each other is the art of mode-locking.

Kerr Lens Mode-Locked lasers operate by passive mode-locking with a fast saturable absorber. The theoretical foundation of these lasers was first developed by Herman Haus in 1975 [17]. Here we will focus on the most popular development of the last ten years of mode-locking, Kerr-Lens Mode-locking using Ti:Sapphire.

In a passively mode-locked cavity the loss in a cavity is lowered by the dynamic absorption of a material or cavity configuration. The more energy there is in the pulse, the less loss the pulse sees. Saturable absorbers are divided up into two categories:

fast saturable absorbers and slow saturable absorbers. Fast saturable absorbers have a transmission response time that is fast compared to the width of the pulse, while slow saturable absorbers have a response time slower than the pulse.

### 1.4.2 Quantum Control

Quantum Control is an oversubscribed term which generally refers to controlling quantum phenomena. In the context of Quantum Computing and Quantum Information Processing, it is a way of performing a weak measurement on a system to gain some information that will allow a better operation to be performed within the coherence time of the quantum system. After the quantum state decoheres the information gained by that measurement is worthless.

In these experiments, Quantum Control is best called Learning-Loop Quantum Control, where a series of experiments are performed to determine which pulse shape will best suit our goals. The loop time is long compared to the decoherence time of the quantum systems so each experiment is independent. The learning-loop algorithm attempts to discover the optimal pulse which illicit the desired response from the quantum system.

An ultrafast laser pulse can populate an excited state in a molecule in essentially zero time compared to the rates of state evolution in molecules. After the initial population event the excited electrons evolve according to the molecule's Hamiltonian. By probing at a series of times we can determine the transition rates from our excited state to subsequent states and eventually the ground state. Before pulse shaping and learning-loop quantum control we had very little ability to influence which state was excited.

To shape an ultrafast laser pulse, a Gaussian pulse is amplified and reflected off of a grating to perform a Fourier transform of the frequency components of the pulse. These frequency components are spread out spatially and a programmable spatial mask called an acusto-optic modulator (AOM) [18] is used to selectively attenuate the amplitude or adjust the phase of the frequency components of the pulse (top of Figure 1-1). When the pulse is reassembled into the time domain via another grating

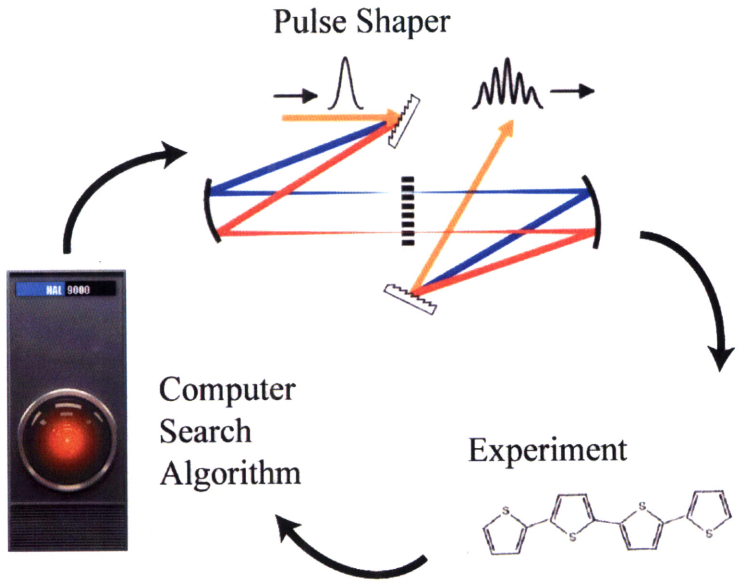



Figure 1-1: Quantum Control Learning Loop: A Pulse Shaper controls the shape of an ultrafast pulse by selectively attenuating frequency components and adjusting phase. The pulse is used in an experiment and the result is fed into a computer algorithm to determine the next pulse.

the temporal envelope has been shaped.

### Molecules

We used the triplet state absorption spectra of molecules as a spatial light modulator in a similar way to the AOM or LCD in current pulse shapers. A thin film placed in the cavity where we switched the molecular elements 'on' using light. The time that they stay 'on' is the duration of their triplet excited state lifetimes. By shining light on a molecular sample we need to illicit a change in the transmission of light around 800nm. One option is to bleach a sample to increase transmission of light. Bleaching is caused by a decrease in the number of molecules able to absorb light when an intense beam is shined on the sample to excite a large percentage of the sample. This excited sample will absorb less light as long as it remains excited.

Typically a molecular sample will be excited from the $S_0$ singlet ground state to the $S_1$ singlet excited state. The transition back down to the ground state is typically very fast. Figure 1-2 shows general energy levels in a molecule. The length of time

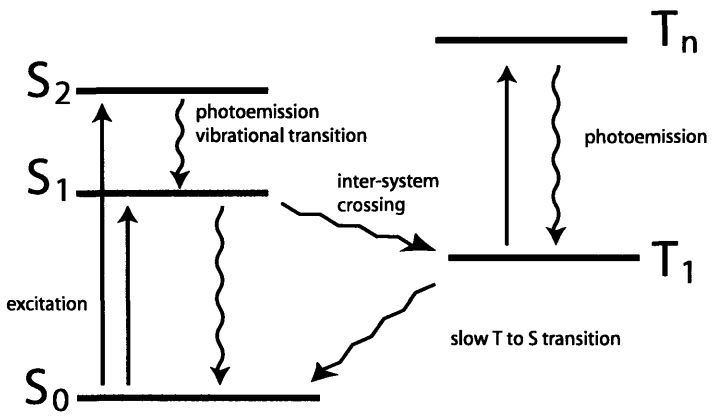



Figure 1-2: General energy levels for a molecule with a triplet state. Light pumps the molecule into the $S_1$ state which can lead to an Inter-System Crossing to the lowest triplet state. A second photon can then excite the molecule into an excited triplet state.

the molecule stays bleached is determined by the rate of $S_1 \rightarrow S_0$ transition. This rate is usually very fast–on the order of picoseconds ($10^{-12}$ seconds).

Alternatively we could select a molecule that is likely to undergo an Inter-System Crossing (ISC) to the Triplet state ($S_1 \rightarrow T_1$) after being excited. Once in the triplet state the transition to the ground state ($T_1 \rightarrow S_0$) is "forbidden" and can take up to milliseconds. Such a molecule would have a long lived 'on' state.

Bleaching only allows us to increase transmission of light going through an optical element. The opposite effect, increasing sample opacity, is also desirable in an optical element.

A molecule excited into a triplet ($T_1$) state can be further excited into higher excited triplet states ($T_1 \rightarrow T_n$) by an additional photon. For the case of a long lived triplet state in a molecule such as Zinc Tetraphenylporphyrin (Figure 1-4) light at 800nm would be absorbed when the molecule has been excited into the lowest triplet state. By pumping this sample at 400nm (Figure 1-3) the singlet absorption will lead to a ISC to the triplet state where the triplet-triplet absorption spectrum extends out to a weak absorption at 800nm. This molecule allows us to turn 'on' absorption. It was an ideal candidate for the triplet spatial light modulator created in Chapter 6.

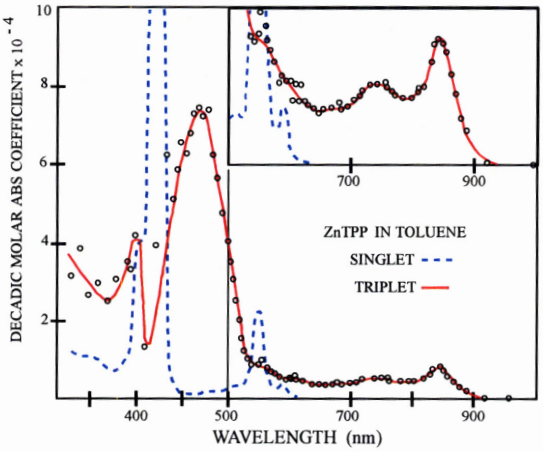



Figure 1-3: The singlet and triplet absorption spectrums of ZnTPP. Figure reproduced from Pekkarinen [4]

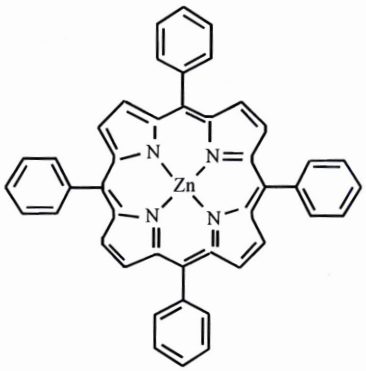



Figure 1-4: ZnTPP molecule

# Chapter 2

# Experiments in Coherent Control of Molecular Triplet State

This chapter reports on our early work searching for shaped ultrafast pulses to enhance the rate of inter-system crossing in metalloporphyrins for NMR polarization enhancement. We found that chirp is the dominant feature in pulses that maximize triplet state population and conclude that the pulses enhanced singlet absorption rather than inter-system crossing.

## 2.1 Introduction

Ultimately our interest in electron spin is motivated by the need to improve Nuclear Magnetic Resonance Quantum Computing (NMRQC). Quantum computers implemented in NMR systems are fundamentally limited by low nuclear spin polarization [19, 14]. The complexity of the problem that can be addressed is constrained by the number of qubits available. Each additional qubit for room temperature liquid NMR has exponentially less signal strength making it difficult to impossible to read the result of the computation. Although the cost can be made polynomial [15], polarization enhancement is necessary to make the prefactors tractable.

Recent innovations in optical pumping of solids [20, 21] have produced polarization enhancements as large as 2.1 x $10^5$. These, and the long history of polarizing

gases [22], inspired us to pursue optical pumping of liquids [13]. Our proposed polarization scheme begins with a singlet excitation followed by an efficient inter-system crossing (ISC) to the triplet state [13]. The triplet electron polarization is transferred to the nucleus by cross-relaxation via a hyperfine coupling. Unfortunately, this polarization scheme can be applied to relatively few molecules. The ISC rate, triplet state lifetime, hyperfine coupling, and applied magnetic field all have to be within narrow ranges to show increased nuclear polarization.

In this chapter we report on our attempts to improve our polarization scheme by increasing the ISC rate of a molecule using shaped ultrafast pulses. Our aim is to expand our list of polarizable molecules to include candidates with naturally inefficient ISCs. We used Zinc Tetraphenylporphyrin (ZnTPP) and Copper (II) Tetraphenylporphyrin for our experiments. These molecules were attractive for their efficient intersystem crossing and long triplet lifetime. Due to the importance of metalloporphyrins in biology these molecules and their triplet properties have been extensively studied [4, 16, 23, 24]. Our coherent control experiments have not succeeded in improving the polarization of the target molecule, but we will describe our methods and suggest possibilities for future control of spin in molecular liquids.

## 2.2 Triplet Polarization Scheme

Here we present one triplet state polarization scheme in liquids. The Hamiltonian evolution the nuclear spin coupled to the triplet state via a hyperfine interaction for the duration of a short lived triplet state, leads to the enhancement of nuclear polarization in a moderate magnetic field ($\approx 1$ Tesla).

In 1974 Bargon and Seifert, reported enhancing nuclear spin polarization by a factor of two in Quinone molecules via a short lived triplet state [25]. The molecules were pumped into the triplet state using a 5kW high pressure mercury-xenon lamp. We assume that the three triplet states are populated in a typical distribution of $(T_-, T_0, T_+) = (10\%, 80\%, 10\%)$. Given a strong hyperfine coupling and a magnetic field near 1.4 Tesla, the rate of polarization transfer from the $T_+$ state to the hydrogen

nuclear spin state was higher than the polarization transfer rate of the $T_-$ state to the hydrogen nuclear spin. This asymmetry is due to the energy difference between the electron Zeeman energy and the hyperfine energy splitting. [26] The optimal ratio of hyperfine coupling $A$ to Zeeman splitting $B_e$ is $A/B_e = 1/2$. At this splitting, the theoretical maximum polarization could reach as high as 41.8% [5]. At the high magnetic fields of commercial NMR spectrometers (11 Tesla), the Zeeman splitting will be far greater than any known hyperfine coupling strengths and the polarization transfer rate will be small (Figure 2-1).

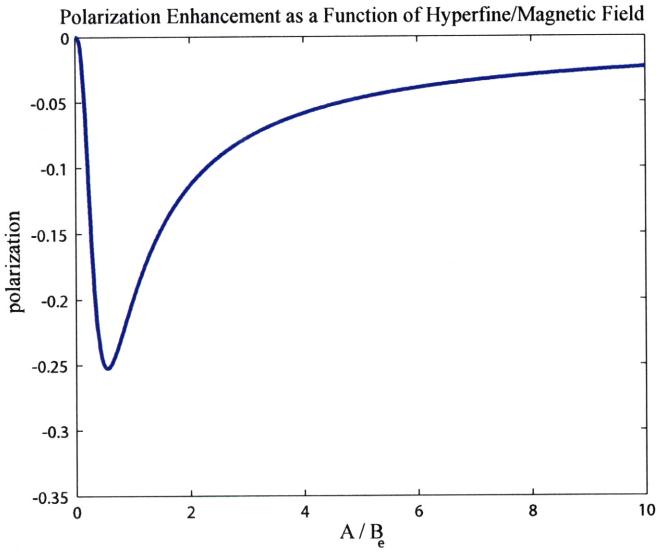



Figure 2-1: Polarization enhancement as a function of the ratio of hyperfine coupling energy to electron Zeeman energy. The maximum enhancement occurs at 0.5. [5]

We believed that pumping with shaped ultrafast pulses could enhance this or a similar polarization scheme.

## 2.3 Experiment

The laser system used in these experiments produces shaped ultrafast light from 100 fs to 5 ps with a spectral bandwidth of 4-5 THz or 12nm around 800nm and up to 1mJ per pulse. As described previously [27], our Ti:Sapphire oscillator produces 100 fs pulses at 790 nm at a repetition rate of 90MHz. The pulses are stretched via a

grating expander to 150 ps and picked by two Pockel cells at 10 Hz. A regenerative amplifier increases the pulse energy to 2mJ where the beam is split into a reference beam, which is compressed for later use, and a beam that is directed into an acousto-optic modulator (AOM) pulse shaper [28]. The AOM modulates the phases and amplitudes of the frequency components of the beam to produce a shaped pulse. The pulse is then amplified in a three-pass amplifier to 1mJ and compressed for use in experiments.

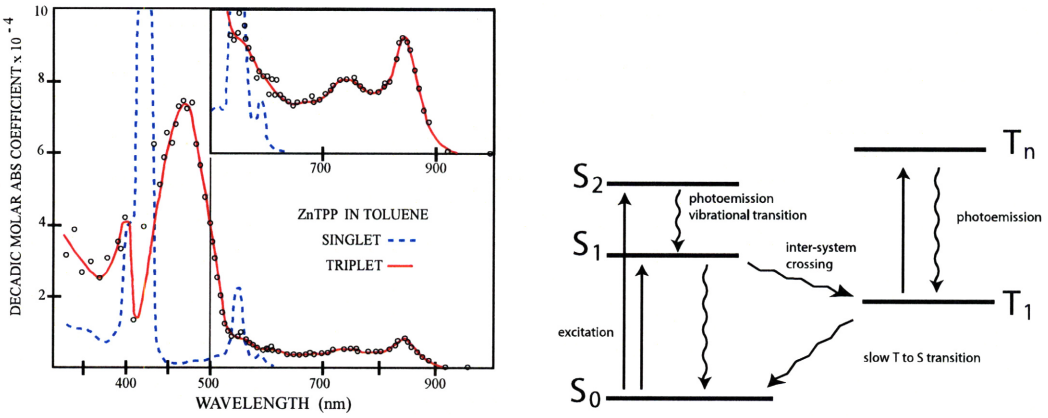



Figure 2-2: The singlet and triplet absorption spectrums of ZnTPP. Figure reproduced from Pekkarinen [4]; Energy levels of metalloporphyrins.

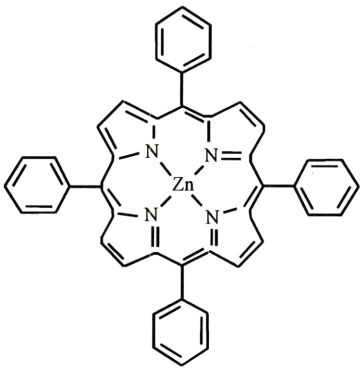



Figure 2-3: Zinc Tetraphenylporphyrin (ZnTPP)

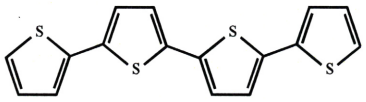



Figure 2-4: Quaterthiophene

In our experiments we used a learning algorithm to search for pulse shapes that would maximize the triplet state population of Zinc Tetraphenylporphyrin and Copper(II) Tetraphenylporphyrin in solution. Metalloporphyrin samples were prepared

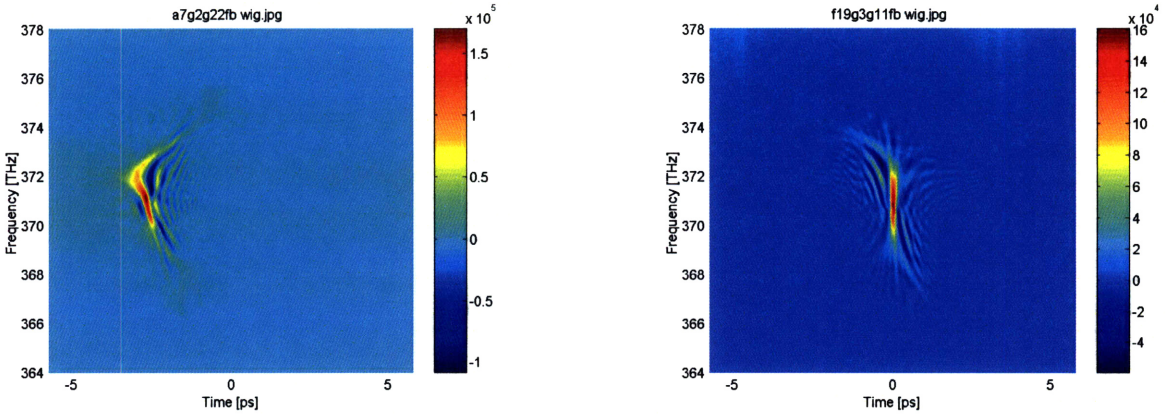



Figure 2-5: Wigner plots of pulse shapes found to minimize (left) and to maximize (right) triplet absorption. Wigner plots display intensity as a function of time and frequency. These plots led us to suspect that chirp was the dominate feature.

in $10^{-4}$ to $10^{-3}$M concentrations in Toluene. Both porphyrins were ordered from Sigma-Aldrich and used without processing. Fresh samples prepared from new bottles of dry 99.8% grade Toluene were used as mixed. Older samples were degassed by several freeze thaw cycles to remove oxygen which quenches the triplet state. We used the shaped pump and unshaped reference in a pump-probe configuration to monitor the triplet state population. The 800nm, 100 fs, reference pulse was brought to a hard focus in a quartz flat to generate a white light continuum [29]. The shaped pulse was doubled in a 1 mm BBO crystal to 400 nm and used to pump the sample. The metalloporphyrin samples absorbed 400nm light in a S0 $\rightarrow$ S2 transition which is followed by a S2 $\rightarrow$ S1 before undergoing an efficient inter-system crossing (ISC) to the triplet ground state ($\phi = 0.88$) which has a long lifetime ($\tau^{1/2} = 2$ms at 298 K) [16]. We monitored the triplet population by integrating the triplet absorption spectra from 430nm to 510nm of the white light probe at pump-probe delays from 5ps to 10ns. We used this measurement of triplet population as figure of merit for the learning algorithm.

The learning algorithm is a modified genetic algorithm (GA) [30] that uses evolution inspired operations such as crossover and mutation as well as physical operators such as smoothing and time-domain crossover [27]. A typical GA run lasted 20 generations before converging. Each generation consisted of 60 individuals with 45 genes

of which 25 to 35 control phase and 10 to 20 control amplitude.

## 2.4 Results and Discussion

Our focus was to increase the rate of inter-system crossing to the triplet state in our molecules. We conducted approximately 140 search experiments. Over the course of our experiments we used a variety fitness functions: absolute triplet state population, triplet state population per unit pump power, and singlet absorption. We found several successful pulse shapes, but the only consistent feature among these pulses was chirp.

Figure 2-5 shows two Wigner plots [31] of pulse shapes found to control triplet state population. These plots led us to suspect that chirp was the dominate feature. The negative chirp pulse shape on the left was discovered when we minimized triplet population while maximizing pump power. A negatively chirped pulse has frequency components that go from high to low as a function of time. The pulse on the right is a transform limited pulse found when we attempted to maximize absolute triplet population.

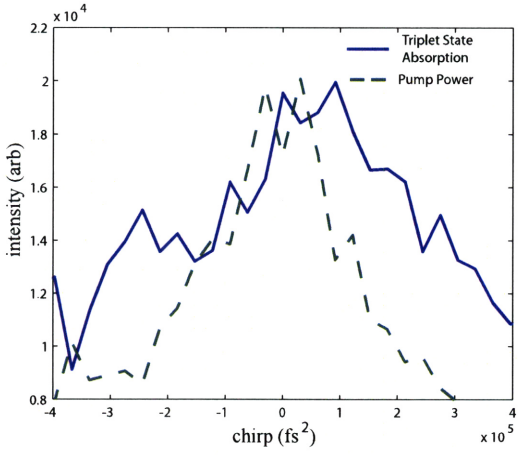



Figure 2-6: Relative intensities of the pump power (green dashed line) and the triplet state absorption (blue solid line)

We monitored triplet state absorption as we scanned our pump from a large negative chirp to a large positive chirp (400,000 $fs^2$). We found clearly less triplet state

population with negatively chirped pulses than positively chirped ones (Figure 2-6). The triplet state population is a product of the pumping efficiency to the excited singlet state and the efficiency of inter-system crossing to the triplet state. The chirped pulses that our GA found enhanced the absorption of light into the singlet state. The magnitude of chirp and the amount of signal enhancement we observed is consistent with the intra-pulse pump-dump process proposed by Shank et. al. [32]. Our shaped pulses were not enhancing the rate of ISC to the triplet state.

The fact that the GA did not find pulse shapes beyond chirped ones to maximize the triplet state population does not mean none exist. The genetic algorithm samples a broad range in pulse shape space but the number of samples is very small ($\approx 1200$) compared to the millions of possible pulse shapes.

We briefly tried additional GA runs using another sample, quaterthiophene (Figure 2-4). The quaterthiophene absorption spectra allowed us to pump at 400nm into the S1 state which directly proceeds the ISC to the triplet state. Our results with quaterthiophene were similar to ZnTPP and CuTPP–no triplet population enhancement beyond the chirp effect.

## 2.5 Conclusions and Future Work

We discovered no new way to enhance triplet state production in molecules. Although, due to the nature of search with a genetic algorithm, we have not eliminated the possibility of a future experiment discovering a pulse shape to increase the rate of inter-system crossing.

The chirp dependence we observed is consistent with the pump-dump process [32]. A negatively chirped pulse first pumps a molecule into an excited state with the leading, higher-energy, portion of the pulse. It then follows with a lower-energy side, corresponding to the Stokes shift of the molecule, which dumps the electrons back to the ground state. A positively chirped pulse counteracts this effect and is most efficient for pumping.

While this work did not lead to improvements in NMR polarization we found the

molecules useful in later experiments.

# Chapter 3

# Kerr Lens Mode-Locked Laser Dynamics

In the previous chapter I assumed a conventional model of computation [33]. However, there are many ways to represent and process information other than digital and quantum computation. Coherent light shown through a disordered medium can implement a cryptographic function [34]. Glow discharge in a microfluidic chip has found shortest paths [35]. Analog logic has been used to implement message passing [36, 37]. Many other examples of physical computation exist [38, 39, 40, 41, 42].

We are interested in physical systems that process information–not to solve computationally hard problems–but to extract a functional description that we can use to refine our use of their dynamics. In this chapter we review the physics of the ultrafast laser used in our coherent control experiments. We implemented a simulation of the oscillator to better understand the process of creating ultrashort pulses. In the chapters to follow we use this simulator to investigate the information processing capacity of these lasers.

Kerr lens mode-locked (KLM) lasers were first realized in 1991 [11]. This laser had a pulse width of 60 femtoseconds. Lasers producing pulses this short are known as ultrafast lasers. At this time scale, most molecular dynamics are frozen. Molecules do not spin, vibrate, or even emit light at a rate faster than these laser pulses. This makes ultrafast lasers a invaluable tool for studying molecular dynamics [12]. Here

we examine how a KLM laser forms ultrashort pulses.

KLM lasers mode-lock when self-focusing in the Kerr medium causes the loss in a cavity to decrease with increasing pulse power. Shorter pulses experience less loss and therefore more net gain. This feedback causes mode-locking of many frequencies in phase with each other resulting in an ultrashort pulse. Using a Ti:Sapphire ($Ti:Al_2O_3$) crystal as both the gain medium and the Kerr medium as been successful and popular. The pulse reaches a steady state solution when the pulse shortening effect of the fast saturable absorber is balanced by pulse lengthing effects such as dispersion or bandwidth filtering. The soliton-like pulse that is formed is a result of a balance between GVD, SPM. Gain saturation, bandwidth filtering, and the pulse shortening effect of the Kerr media stabilize the pulse. We will review these soliton effects, mode-locking, and KLM cavities.

## 3.1 Mode-locking

The term 'Mode-locking' refers to the frequency modes in a laser cavity. Typically we think of a short laser pulse in the time domain. A Gaussian laser pulse in the time domain is also a Gaussian shape in the frequency domain. Constructing a laser cavity to excite nearby cavity modes in phase with each other is the art of mode-locking.

Kerr Lens Mode-Locked lasers operate by passive mode-locking with a fast saturable absorber. The theoretical foundation of these lasers was first developed by Herman Haus in 1975 [17]. The science of short laser pulses lasers has progressed steadily over the last 30 years. Figure 3-1 shows progress by pulse width, year, and material. This chapter will focus on the most popular development of the last ten years of mode-locking, Kerr-Lens Mode-locking using Ti:Sapphire.

Mode-locking can be divided up into two broad categories: active mode-locking and passive mode-locking. Active mode-locking of a cavity is driven by electronics and passive mode-locking is an all-optical process. Passive mode-locking, having the advantage of requiring no electronic parts, is not limited by current GHz electronics. Passive mode-locking provided for several orders of magnitude improvement in the

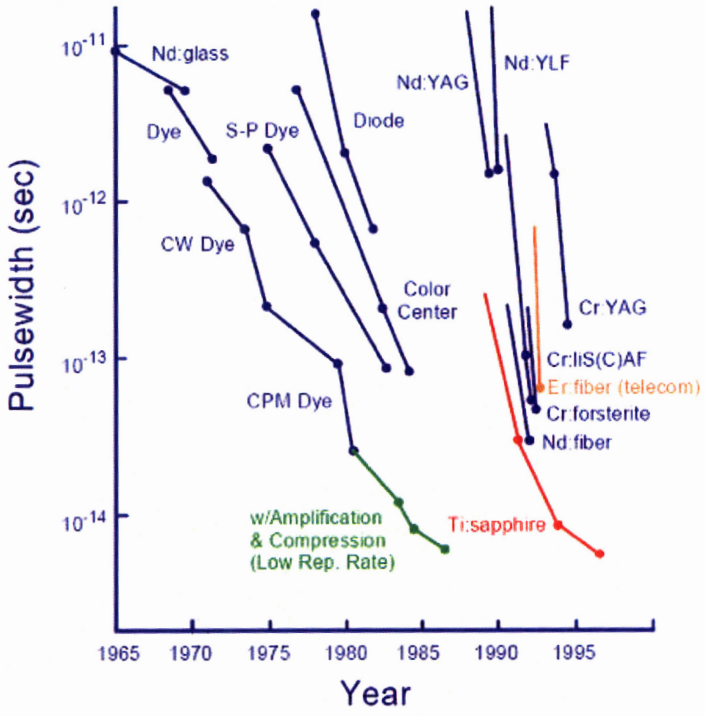



Figure 3-1: Pulsewidths by year and material [6].

shortness of the pulse.

Siegman and Kuizenga developed the analytic theory of an actively mode-locked laser in 1970 [43, 44]. An actively mode-locked cavity contains an electronically controlled element such as an electro-optical crystal or acousto optic modulator. This introduces a large loss for all modes except the ones to excite. By driving the amplitude modulator to high transmission at the round-trip time of the cavity short pulses are produced. By favoring a single pulse circulating in the cavity successive passes through the gain medium excites nearby cavity modes to shorten the pulse. Nearby modes modes exist with mode spacing of $\Delta\omega = 2\pi c/2L$, where $L$ is the cavity length.

In the frequency domain, individual cavity modes experience more gain if nearby modes are already excited [6].

$$\Delta A_n = \left[\frac{g}{1+(\frac{n\Delta\Omega}{\Omega_g})^2} - l\right] A_n + \frac{1}{2}m(A_{n-1} - 2A_n + A_{n+1}) \tag{3.1}$$

Equation (3.1) shows the change in mode amplitude $A_n$ as a function of mode $n$, gain

$g$, loss $l$,

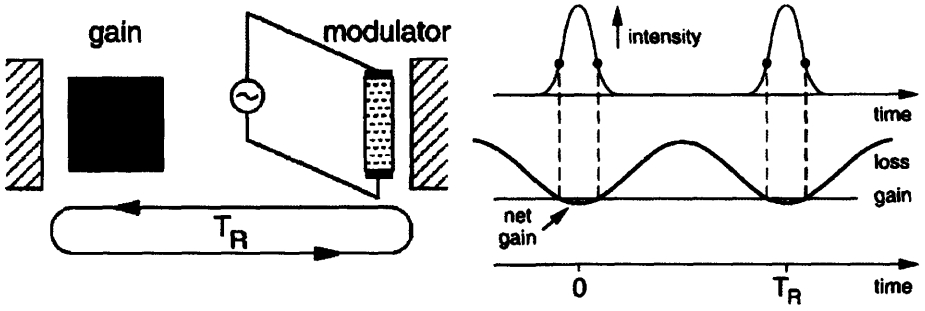



Figure 3-2: Active Mode Locking [6].

If we assume a Gaussian-like pulse we can obtain a Pulse-Shortening Rate (PSR) for active mode-locking [45]

$$\frac{\Delta\tau}{\tau} = \frac{m\omega_m^2\tau^2}{4}$$

where $\tau$ is our pulse width and $\Delta\tau$ is a decrease in pulse width. We can quickly see that as the pulse width $\tau$ gets smaller, the effectiveness of pulse shortening goes down rapidly ($\tau^2$). Passive mode locking using a slow saturable absorber is much more favorable for generating short pulses.

In a passively mode-locked cavity the loss in a cavity is lowered by the dynamic absorption of a material or cavity configuration. The more energy there is in the pulse, the less loss the pulse sees. Saturable absorbers are divided up into two categories: fast saturable absorbers and slow saturable absorbers. Fast saturable absorbers have a transmission response time that is fast compared to the width of the pulse, while slow saturable absorbers have a response time slower than the pulse. Typically slow saturable absorber lasers use a combination of loss saturation and gain depletion to obtain short pulses.

Slow saturable absorbers have a decreasing absorption when subjected to incident radiation. This effect is often called bleaching. Short pulses can be made using a saturable absorber and a saturable amplifier. Figure 3-3 shows a slow saturable absorber laser cavity. The loss in the cavity decreases over the width of the pulse and recovers slowly but within the round-trip time ($T_R$) of the cavity. Simultaneously the gain medium is depleted of population inversion to form the trailing edge of the

pulse.

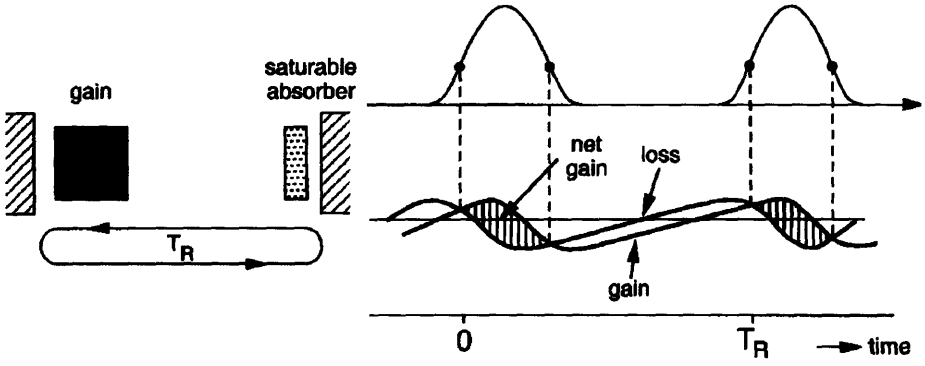



Figure 3-3: Slow Saturable Absorber [6]

The saturation of the gain medium can be described as [6]

$$g(t) = g_i \exp\left(-\int_0^t dt \frac{|a|^2}{W_g}\right),$$

where $g_i$ is the gain before the pulse arrives. The gain decreases with increasing time over a pulse. Similarly the saturable absorption can be described:

$$s(s) = s_i \exp\left(-\int_0^t dt \frac{|a|^2}{W_s}\right)$$

where $s_i$ is the saturable loss before the pulse arrives.

The pulse shortening per pass is given by [45]:

$$\frac{\Delta\tau}{\tau} = \text{const.}$$

Unlike active mode-locking, the pulse shortening rate of passive mode-locking with a slow saturable absorber is constant over all pulse widths. The shortening does not slow down with fast pulses.

## 3.2 Fast Saturable Absorber Mode-locking

A fast saturable absorber has an absorption recovery time that is fast compared to the width of the pulse. Figure 3-4 shows the recovery of the absorption. Unlike the slow saturable absorber we assume that the gain is constant over the duration of the pulse. The fast recovery of the absorption shortens the trailing edge of the pulse.

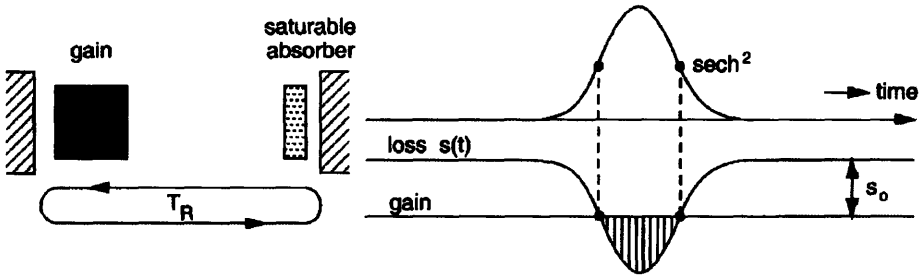



Figure 3-4: Fast Saturable Absorber [6]

The fast saturable absorber is modeled by

$$s(t) = \frac{s_0}{1 + I(t)/I_{\text{sat}}}$$

where $s_0$ is the unsaturated loss, $I(t)$ is the intensity of the pulse, and $I_{\text{sat}}$ is the saturation intensity of the absorber.

Following Haus [6], if the saturation is weak we can expand this to

$$s(t) = s_0 - s_0 \frac{I(t)}{I_{\text{sat}}}.$$

If we normalize $|a(t)|^2|$ to be power then we can write

$$s(t) = s_0 - \frac{s_0 |a(t)|^2}{I_{\text{sat}} A_{\text{eff}}} \equiv s_0 - \gamma |a(t)|^2$$

where $\gamma$ is the self amplitude modulation (SAM) coefficient.

Haus' master equation balances gain, loss, gain bandwidth filtering, and the saturable absorber.

$$\frac{1}{T_R}\frac{\partial}{\partial T}a = (g-l)a + \frac{g}{\Omega_g^2}\frac{\partial^2}{\partial t^2}a - (s_0 - \gamma|a|^2)a \tag{3.2}$$

We can combine $s_0$ into $l$

$$\frac{1}{T_R}\frac{\partial}{\partial T}a = (g-l)a + \frac{g}{\Omega_g^2}\frac{\partial^2}{\partial t^2}a + \gamma|a|^2 a. \tag{3.3}$$

Later we will expand gain bandwidth broadening term, $g/\Omega_g^2$, to include cavity bandwidth filtering and call these dispersive effects $D_{gf}$ where $D_{gf} = g/\Omega_g^2 + 1/\Omega_f$.

The solution to this master equation (3.3) is

$$a_0(t) = A_0 \operatorname{sech}(t/\tau), \tag{3.4}$$

where

$$\frac{1}{\tau^2} = \frac{\gamma A_0^2 \Omega_g^2 a}{2g}$$

and

$$l - g = \frac{g}{\Omega_g^2 \tau^2} \tag{3.5}$$

By inspection of (3.3) and (3.4) we can see that this analytical solution is unbounded. Gain saturation must be introduced into the model to arrive at a stable pulse. Equation 3.5 is also a bit confusing at first glance. Typically for laser operation we would assume that gain $g$ would be larger than loss $l$ making Equation 3.5 false. This equation applies for noise in the cavity during pulsed operation. We have negative net gain for noise while the pulse by way of Self Amplitude Modulation sees less loss through the saturable absorber and therefore positive net gain.

The pulse shortening per pass is given by [45]:

$$\frac{\Delta\tau}{\tau} = \frac{\gamma W}{2\tau} \tag{3.6}$$

The PSR given in (3.6) shows an increase in pulse shortening with shorter pulses. Recall pulse shortening in active mode-locking became weaker ($\tau^2$) with shorter pulses. In passive mode-locking with a slow saturable absorber the pulse shorten-

ing rate was independent of pulse width. For a fast saturable absorber the pulse shortening rate increases as $\tau$ gets smaller $(1/\tau)$. We will find that for a fast saturable absorber the evolution of our ultrashort pulses will be determined by soliton effects.

This inverse-$\tau$ relationship has one drawback. For long pulses the pulse shortening rate is small. This is the primary reason most fast saturable mode-locked lasers are not self-starting and need to be started by perturbing the cavity in some way. A short pulse is necessary to get the pulse shortening started.

### 3.2.1 Kerr Lens Mode-locking

Kerr lens mode-locked lasers operate by an artificial fast saturable absorber caused by the combination of Kerr lensing and an aperture to give a near instantaneous response time much faster than pulse widths.

The Kerr effect causes the refractive index of a medium to increase with intensity:

$$n(I) = n_0 + n_2 I$$

where

$$n_0 = 1.76,\ n_2 = 3 \text{ x } 10^{-20}\ m^2 W^{-1}$$

for Ti:Sapphire.

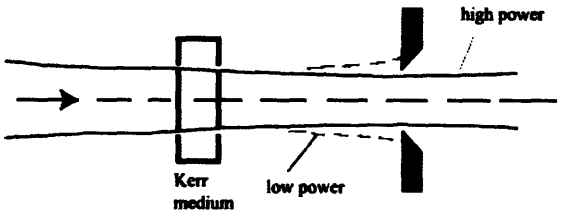



Figure 3-5: Kerr medium and aperture.

We assume the spatial mode of the pulse is Gaussian. The center of the pulse has a higher intensity than the outer part. When the pulse goes through the Kerr medium in the cavity, the center sees a greater index of refraction than the outer part. This

causes a focusing that is dependent on total pulse intensity. By putting an aperture in the cavity that is smaller than the CW pulse width we can encourage shorter more intense pulses (Fig. 3-5). The net effect of the Kerr media and the aperture creates a fast saturable absorber. ABCD-matrix beam propagation can be used to describe this self-focusing [46].

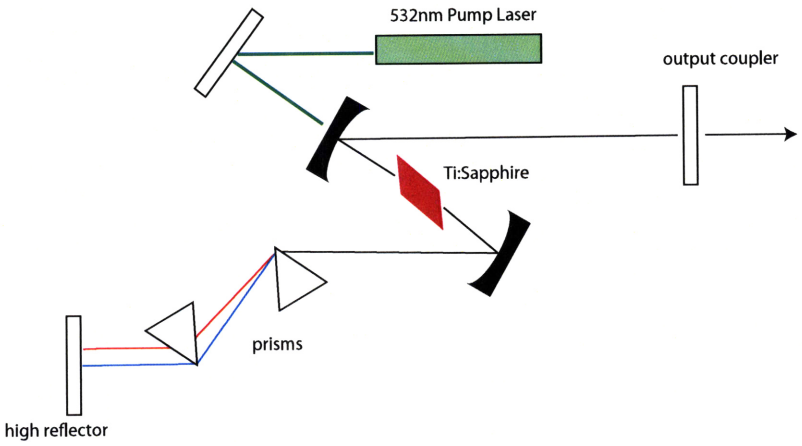



Figure 3-6: KLM Cavity

The Ti:Sapphire KLM oscillator design from 1991 has remained largely unchanged. Figure 3-6 shows the Ti:Sapphire crystal pumped at 532nm. The Titanium doped Sapphire crystal acts as both the gain medium and the Kerr medium. The Kerr focusing in combination with either a real aperture, cavity misalignment, or pump/cavity beam overlap forms the artificial fast saturable absorber. The two prisms are included to compensate for the positive group velocity dispersion (GVD) caused during propagation through the Ti:Sapphire crystal. More recent cavity designs use chirped mirrors to compensate for Third Order Dispersion (TOD) and GVD rather than prisms. An acusto-optic modulator is sometimes added in the cavity to initiate mode-locking.

## 3.3 Soliton Effects

Ultrashort pulses propagating in a medium typically experience Group Velocity Dispersion. The broad bandwidth frequency components of a transform limited pulse experience an index of refraction based on their frequency $n(\omega)$. This causes the red side of the pulse to get ahead of the blue side of the pulse in the case of normal dispersion. This is called a positive chirp. In the case of anomalous dispersion the blue side of the pulse propagates faster than the red resulting in a negatively chirped pulse. This chirping causes the pulse to become broader in time but the frequency spectrum remains unchanged.

If we can apply group velocity dispersion (GVD) to our pulse

$$\Delta a = jD\frac{d^2}{dt^2}a.$$

Most materials have normal dispersion which induce a positive chirp. In a KLM cavity it is essential to compensate for this positive chirp with negative dispersion. Negative group velocity dispersion can be made using a pair of prisms [47] or diffraction gratings [48]. This balance between the positive dispersion of the cavity elements and the negative dispersion of the prism configuration allows for near zero net dispersion.

Self-Phase Modulation (SPM) changes the phase of a pulse according to the instantaneous pulse intensity. In a Kerr medium the phase will be delayed more for higher intensity.

$$\Delta a = -j\delta|a|^2 a$$

$$n(I) = n_0 + n_2 I$$

where $n_0 = 1.76$ and $n_2 = 3$ x $10^{-20}$ $m^2W^{-1}$ for Ti:Sapphire.

Our full master equation is

$$\frac{1}{T_R}\frac{\partial}{\partial T}a = (g-l)a + \left(\frac{g}{\Omega_g^2} + \frac{1}{\Omega_f^2} + jD\right)\frac{\partial^2}{\partial t^2}a + (\gamma - j\delta)|a|^2 a \tag{3.7}$$

where $g$ is gain, $l$ is loss, $\Omega_g$ is gain filtering, $\Omega_f$ is cavity filtering, $D$ is the GVD parameter, $\gamma$ is the SAM coefficient, $\delta$ is the SPM coefficient.

## 3.4 Numerical Simulation of a Mode-Locked Oscillator

We use the master equation approach [17, 6, 10, 9] to simulate the dynamics of a KLM oscillator. The master equation describes passive mode-locking using fast saturable absorbers and includes Group Velocity Dispersion (GVD), Self-Amplitude Modulation (SAM), Self-Phase Modulation (SPM), and bandwidth filtering. Most oscillator dynamics that are not explicitly dependent on spatial modes can be included in this model.

Our simulations are seeded with a Gaussian pulse in the picosecond to 100 femtosecond range. The master equation governs the pulse evolution simulating one cavity round-trip of the pulse for each time step. Typical simulations converge on a steady state solution within 2,000 to 8,000 round trips. The non-linear and dispersive effects were applied to the pulse using the Split-Step Fourier Transform method [49]. This numerical method is commonly used to simulate intense pulses propagating in optical fiber.

In developing this simulation we began by reproducing a simulation of a Nd:glass laser [9]. We then adapted this simulation to use Ti:Sapphire parameters. At the end of this chapter we will present the results of both simulations. First we review the master equation including soliton effects for a KLM cavity and examine the Split-Step Fourier Transform Method. We then present simulator implementation details and finally present evidence demonstrating the accuracy of our simulator.

## 3.5 Split-Step Fourier Method

At the core of this simulation is the split-step Fourier method.

Here we review Agrawal's derivation [49] of the split step Fourier method (SSFM). We've adapted his use of the SSFM in pulse propagation in optical fiber to our master equation describing a KLM cavity. In the fiber derivation, the length of the fiber is broken down into a large number of segments and then the dispersive terms and nonlinear terms are applied separately to each segment. If we view our master equation in the form,

$$\frac{\partial A}{\partial T} = (\hat{D} + \hat{N})A, \tag{3.8}$$

we have linear terms,

$$\hat{D} = g - l + (D_{gf} + jD)\frac{\partial^2}{\partial t^2}, \tag{3.9}$$

and the nonlinear terms,

$$\hat{N} = (\gamma - j\delta)|A|^2. \tag{3.10}$$

We assume that the dispersive terms and non-linear terms can be applied to the pulse separately. We also assume that effects of one round trip on a pulse are small so we can apply the operators in one step rather than breaking up the cavity into several segments.

The exact solution to (3.8) is

$$A(T + \kappa, t) = \exp[\kappa(\hat{D} + \hat{N})]A(T, t)$$

where $\kappa$ is a small time interval.

The split-step Fourier transform method approximates this solution as

$$A(T + \kappa, t) \approx \exp(\kappa\hat{D}) \exp(\kappa\hat{N})A(T, t). \tag{3.11}$$

Note that operators $\hat{N}$ and $\hat{D}$ do not commute. The higher order terms dropped in equation (3.11) can be found using the Baker-Campbell-Hausdorff formula.

$$\exp(\hat{a})\exp(\hat{b}) = \exp(\hat{a} + \hat{b} + \frac{1}{2}[\hat{a}, \hat{b}] + ...)$$

The approximation is valid assuming higher order terms beginning with

$$\frac{1}{2}\kappa^2[\hat{D}, \hat{N}], \tag{3.12}$$

and above can be ignored for small $\kappa$. Since we expect the formation of pulses in our laser cavity to take many hundreds of round trips with small changes per iteration. We can assume that our analogous $\kappa$ can be considered small and that the approximation in (3.11) is appropriate.

A further refinement to this method, called the symmetrized split-step Fourier method, uses:

$$A(T + \kappa, t) \approx \exp\Big(\frac{\kappa}{2}\hat{D}\Big)\exp\Big(\int_T^{T+\kappa} \hat{N}(T')dT'\Big)\exp\Big(\frac{\kappa}{2}\hat{D}\Big)A(T, t) \tag{3.13}$$

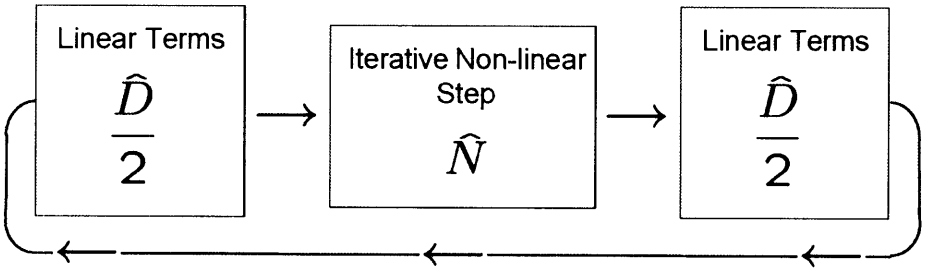



Figure 3-7: Symmetrized Split-Step Fourier Transform Method

Using this symmetrized SSFM the higher order terms we dropped go to zero as $\kappa^3$ rather than $\kappa^2$ (3.12). Since $\hat{D}$ is linear it can be split into two steps.

The final steps in Agrawal derivation comes down to the integral over a non-linear operator.

$$\int_T^{T+\kappa} \hat{N}(T')dT' \approx \frac{h}{2}[\hat{N}(T) + \hat{N}(T + \kappa)]$$

Unfortunately we don't know $N(T+\kappa)$ when we are calculating $A(T+\kappa)$ since $N(T+\kappa) = f[A(T+\kappa)]$. We will have to use an iterative method where we first assign $N(T+\kappa) = N(T)$ in order to calculate a first approximation to $A(T+\kappa)$ then we use that value for our second iteration using our new $N(T+\kappa)$ to update our estimate of $A(T+\kappa)$. We continue this iteration until successive values of $A$ are within a specified tolerance.

At the onset of writing this simulator, it wasn't clear if iterating the non-linear step was necessary. Rarely did our number of iterations exceed 3–typically only two iterations occurred. Future work on this simulator should remove the iterative step to test simulator robustness.

### 3.5.1 Simulation Procedure

The simulator is a relatively simple implementation of the pulse evolution described in Haus' master equation (3.7). The split-step Fourier transform method was used to evolve a pulse through one round trip of cavity. We apply the linear cavity effects in the frequency domain and we apply the non-linear saturation and phase modulation effects in the time domain.

We seed our simulations with a relatively broad bandwidth low intensity pulse. In physical KLM oscillators it is usually necessary to perturb the cavity by moving one of the dispersion compensating prisms in order to seed the cavity with something pulse-like to feedback on. Fast saturable absorber oscillators are typically not self starting since the pulse shortening rate is weak for long pulses and successively stronger for short pulses.

We assume that that the population lifetime of our gain medium is long compared to the repetition rate of the cavity. This means that the gain will saturate according to the average power over many pulses rather than over a single pulse width. To accommodate this assumption we calculate the effective gain between cavity round trips. In each of the cavity round trip iterations we calculate a new gain $g$ according

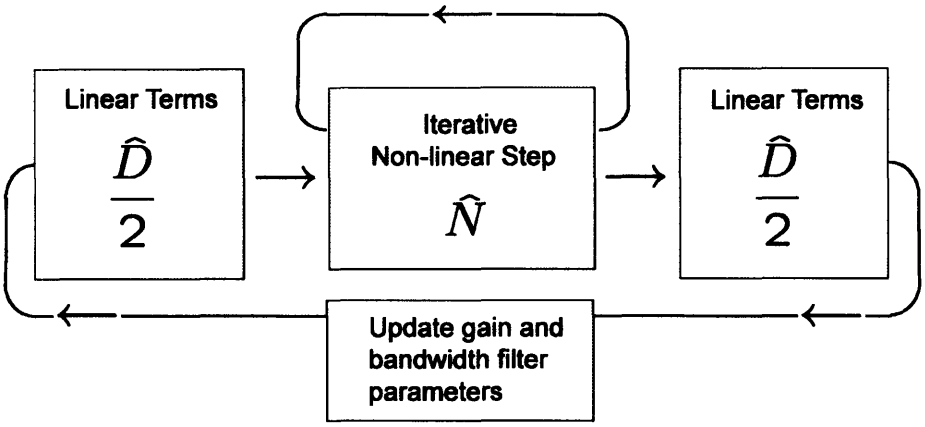



Figure 3-8: Simulator Block Diagram

to:

$$g(T) = \frac{g_0}{1 + \frac{E_P(T)}{E_{sat}}}$$

where

$$\begin{aligned} E_P(T) &= \int_{-\infty}^{+\infty} |A(T,t)|^2 dt \\ E_P(T) &= \Delta t \sum_{n=0}^{N} |A(T,n)|^2 \end{aligned}$$

Dispersion due to gain bandwidth and cavity bandwidth is also updated

$$D_{gf} = g/\Omega_g^2 + 1/\Omega_f^2$$

### Pseudo Code

Here is the pseudo code for one round trip time in the cavity. A minimum of 4 Fourier transforms are performed. Typically the iterative step converges within two iterations. Most simulations performed around 30,000 Fourier transforms while converging on a pulse.

```
for N = 1 to ( number of simulation steps (round trips), typically 8,000 )

begin

  calculate gain with depletion

  calculate D_gf coefficient

  prepare linear half step in Fourier domain
  apply linear half step in Fourier domain to the fft of the waveform (HALFSTEP)

  iterate to find non-linear part ->

  for i = 1 to ( max allowed iterations, typically 10 )
     begin

     apply non-linear operator to HALFSTEP the time domain
     multiply result by HALFSTEP in the frequency domain
     return result to time domain

     check to see if successive iterations are within tolerance if not then repeat

     end

  if (over max iterations) stop and report error to user

end
```

## 3.6 Simulations

We began our simulations using parameters for Nd:glass found in a Mode-Locking paper by Kaertner et. al. [9]. The parameters used are listed in Table 3.1. Our first

| Parameter | Value | Description |
|---|---|---|
| $l$ | 0.01 | loss |
| $g_0$ | 0 - 0.2 | unsaturated gain |
| $\Omega_g$ | $2\pi \cdot 4$ THz | gain bandwidth |
| $q_0$ | 0.005 | $s_0$ unsaturated absorber loss |
| $D$ | -75 fs$^2$ | dispersion |

Table 3.1: Table of simulator parameters for Nd:glass [9].

simulations included gain depletion, Self Amplitude Modulation (SAM), and bandwidth filtering only. Self Phase Modulation (SPM) and Group Velocity Dispersion (GVD) were set to zero. Later we switched to Ti:Sapphire parameters and included soliton effects.

Because a typical simulation requires around 30,000 FFTs we tried to keep the number of points in the waveform limited. Since a typical pulse evolution for a KLM oscillator may have a pulse width from 10's of picoseconds down to 10's of femtoseconds we have had to be wary of using too few points in our waveform. Typically we use around $2^{13} = 8192$ points in our array.

### 3.6.1 Nd:glass

We worked out most of the simulator bugs by reproducing the simulation of a saturable absorber Nd:glass laser, the worst of which had to do with our assumptions on the SAM coefficient.

$$s(t) \equiv s_0 - \gamma |a(t)|^2$$

Figure 3-9 shows the pulse evolution over 8000 cavity round trips. For this simulation we used artificially heavy bandwidth filtering to limit the shortness of the pulse. Our simple model of SAM using only the $\gamma$ parameter tended to blow-up since the product $\gamma |a(T,t)|^2$ could grow larger than $s_0$. The derivation of the master equation assumes that it remains relatively small. We will fix this in the model later in this chapter. At the time, we imposed artificially high heavy bandwidth filtering to stabilize the model. It was possible to find SAM coefficients and a value for gain

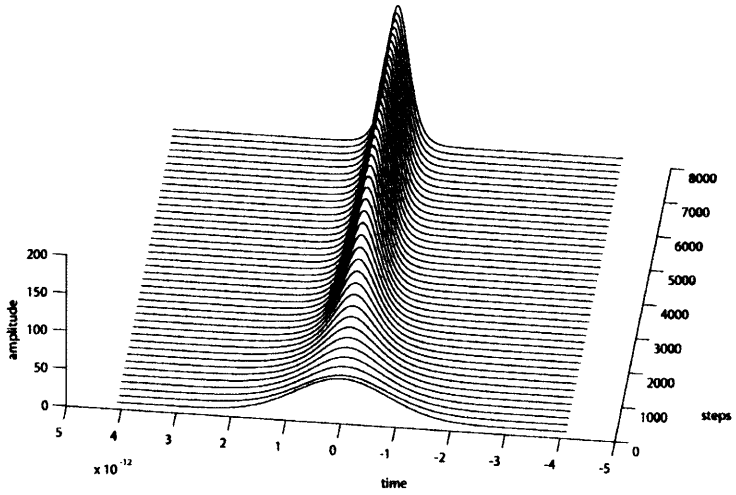



Figure 3-9: job9: Nd:glass, Pulse ring-up over 8000 cavity round trips.

saturation that kept the model stable but it was very difficult.

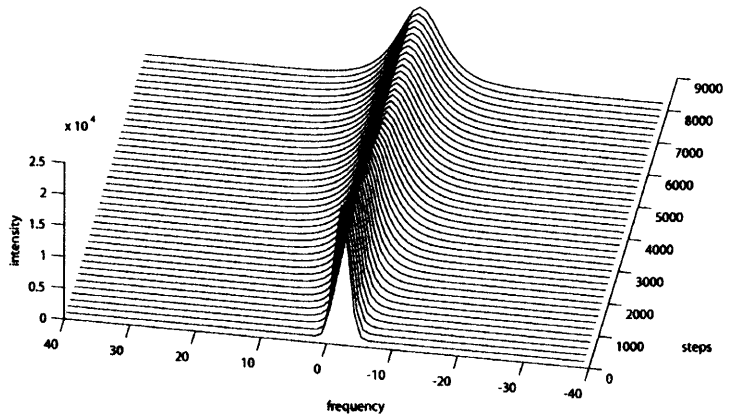



Figure 3-10: job9: Nd:glass, Pulse bandwidth evolution over 8000 cavity round trips

Figure 3-10 shows frequency components of the same series of pulses. No SPM or GVD was included. The converged upon waveforms from figures 3-9 and 3-10 reflect the bandwidth filter we imposed. Figure 3-9 displays an interesting model attribute. Self Amplitude Modulation (SAM) is relatively weak during the first 1,000 or 2,000 steps of the simulation. At around step 3,000 the SAM seems to turn on and shorten the pulse width. For this simulation the shortening was limited by the bandwidth filtering $\Omega_f = 2 * \pi * 0.16$ x $10^{14}$ applied to the system.

The blow-up that occurred was usually a result of either too few points use in the

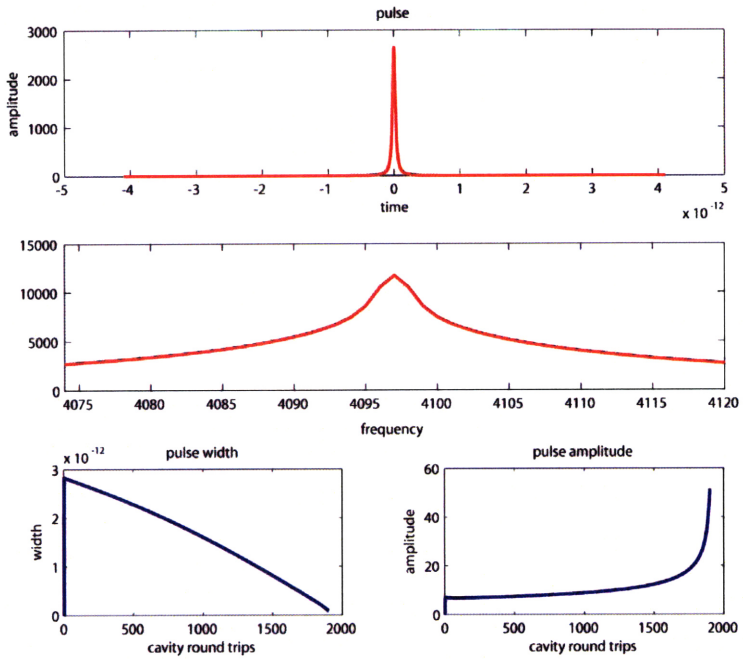



Figure 3-11: job10: Nd:glass, Poor SAM parameter blow-up

waveform or was due to our calculation SAM that continued to give more gain to a pulse even after $\gamma|a(T,t)|^2$ was larger than $s_0$. We fixed this, as Kaertner did, by cutting off gain due to SAM at $s_0$. A typical failed run is shown in Figure 3-11. The pulse length goes to zero and we run out of bandwidth.

| Parameter | Value | Description |
|---|---|---|
| $l$ | 0.025 | loss |
| $g_0$ | 0.0 - 0.10 | unsaturated gain (0.07 typical) |
| $\Omega_g$ | $2\pi \cdot 43$ THz | gain bandwidth |
| $q_0$ | 0.01 | $s_0$ unsaturated absorber loss |

Table 3.2: Table of simulator parameters for Ti:Sapphire [10].

### 3.6.2 Ti:Sapphire

We calculated our Ti:Sapphire model parameters assuming a 50fs pulse, 400mW ML output power, 90MHz repetition rate, and a 3% output coupler. We used a 14W intracavity power or 1.55 x $10^{-7}$ J/pulse corresponding to a pulse amplitude of $A_0 = 1250$ assuming a sech pulse.

$$a_0(t) = A_0 \text{ sech}(t/\tau),$$

Our simulation should be sensitive to our initial gain parameter $g_0$. Below some threshold our net gain should be negative and above some turn-on threshold we should see a net gain. Figure 3-12 shows the final pulse from 20 simulations at gain values from $g_0 = 0$ to $g_0 = 0.2$.

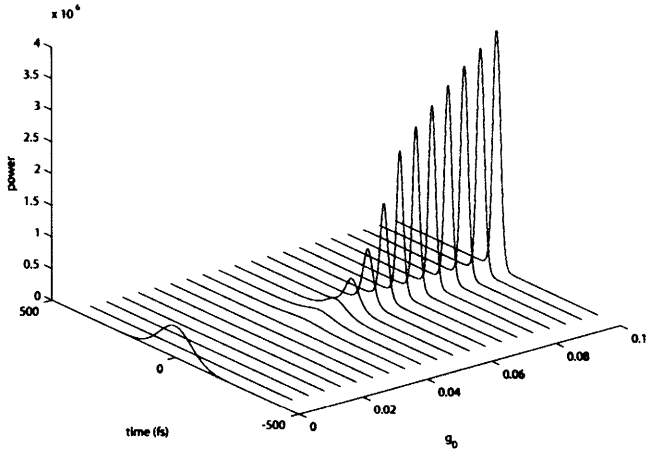



Figure 3-12: job24: Ti:Sapphire, Converged pulse after 8000 iterations for unsaturated gain values from 0 to 0.10, no SPM, no GVD

We can see the net gain turning positive at around $g0 = 0.05$. This threshold is consistent with Figure 3 in Kaertner [9]. Each trace represents 8,000 iterations of the master equation. All pulses converged to a steady state value within this time. Last year's Matlab computer took about 5 minutes to run each simulation for a given $g0$.

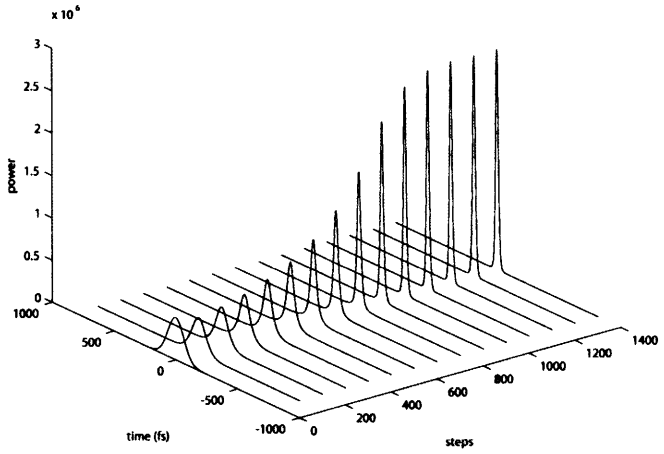



Figure 3-13: job40: Ti:Sapphire, Pulse evolution of an ultrashort pulse.

Group velocity dispersion and self phase modulation were added to the simulation. To view the pulse evolution we saved traces every 100 simulation steps. The evolution of an ultrafast pulse is shown in Figure 3-13. Our initial pulse seed pulse is shown at step 0. For this simulation we seeded the simulation with a relatively short 200fs pulse. The simulations converged to their steady state pulse regardless of initial pulse width or intensity. The final pulse had a width of 38fs. This simulation converged to a steady state solution after only 1500 cavity round trips. We ran the simulation for a few thousand iterations beyond this point to confirm convergence.

The evolution of the frequency components of these pulses is shown in Figure 3-14. Again, the initial pulse is at step 0. The pulse evolution is easily divided up into three stages. Initially, for the first 300 steps, unsaturated gain is the dominant effect. The net gain anything other than an intense pulse is negative. Gain is followed up by self amplitude modulation shortening the pulse. The gain is eventually saturated and the self amplitude modulation effect reaches its minimum saturable loss. At around step

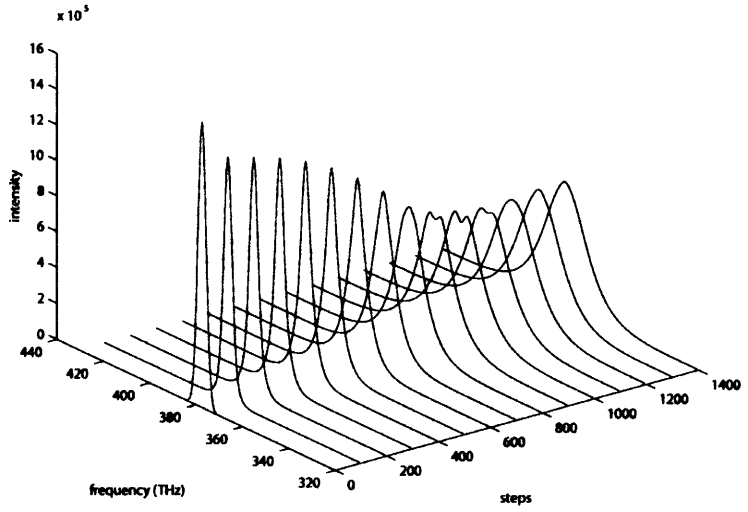



Figure 3-14: job40: Ti:Sapphire, Bandwidth evolution of an ultrashort pulse.

800 soliton effects further shorten the pulse. The dip in the middle of the pulse at step 1000 is due to self phase modulation that is eventually removed by group velocity dispersion.

Our final confirmation of the accuracy of our simulation to theory is a large simulation run intended to duplicate analytical results presented in Figure 8 of review paper by Haus [6]. He calculated normalized pulse width as a function of SPM parameter $\delta$ and GVD $D$. We have reproduced this figure from his equations (fig. 3-15).

$$\frac{\tau}{\tau_0} = -(3/2)(s^2\gamma_n - sD_n) + \sqrt{[(3/2)(s^2\gamma_n - sD_n)]^2 + 2s^2}$$

where

$$s \equiv \frac{1 + D_n^2}{D_n\gamma_n + \delta_n}$$

and $D_n = D\Omega_f^2$, $\gamma_n = W\Omega_f^2\tau_0\gamma/6$, $\delta_n = W\Omega_f^2\tau_0\delta/6$. We see that the shortest pulses should be obtained for negative dispersion and some self phase modulation. This isn't a large increase in pulse shortening.

We performed four sets of simulations at four different values for $\delta$. Each line in Figure 3-16 is made of 20 points taken at evenly spaced values of dispersion for each SPM value. Each point is made from the final pulse after 8,000 simulated cavity

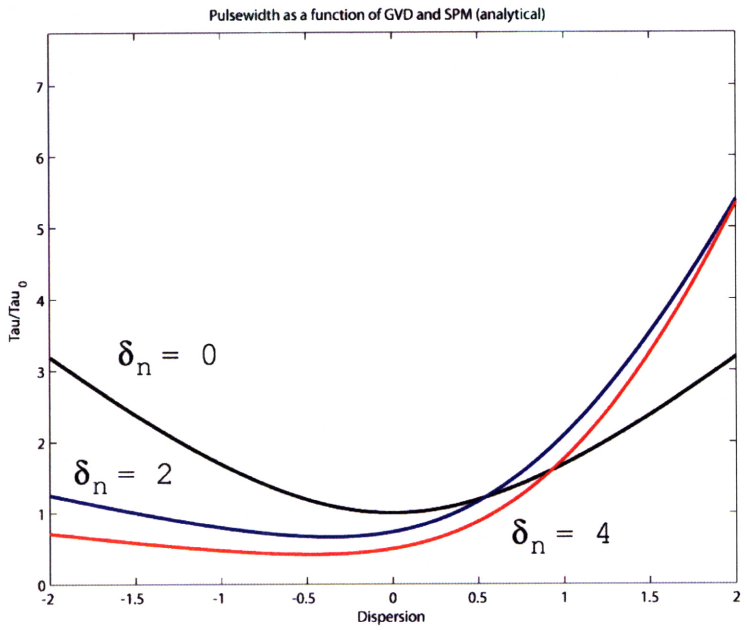



Figure 3-15: Figure reproduced from [6]

round-trips for given GVD and SPM values. Each line took approximately 7 hours of simulation time on desktops from 2002. There were a number of obvious short-cuts to reduce this time that we decided not to include to ensure model accuracy.

All key attributes of the output replicated the Haus analytical result. Our $\delta_n$ and dispersion numbers were based on Ti:Sapphire parameters. The simulation worked very well.

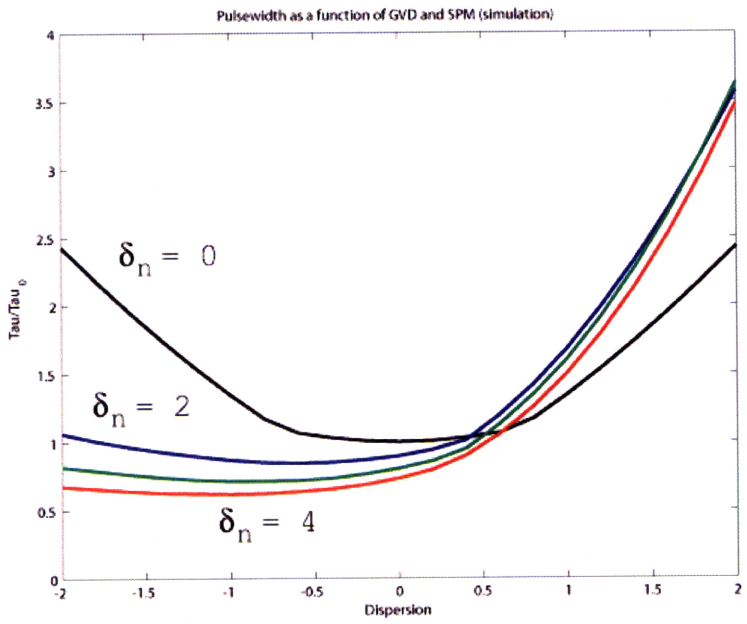



Figure 3-16: job3x: Ti:Sapphire, Simulation data replicating Haus figure

# Chapter 4

# Optical Feedback Shift Registers

In this chapter we use an Analog Feedback Shifted Register (AFSR) [50, 36, 7] as a test problem to investigate the suitability of a Kerr-lens Mode-locked (KLM) laser cavity in a feedback configuration to solve mathematical programs. We have chosen a amplitude and pulse width representation for our bits. Using feedback into the mode-locked cavity we expect the cavity to "find" solutions to the feedback loss that minimize loss. Two such cavities coupled together can be understood as solving a recursive state estimation problem.

We attempt to constrain the physical dynamics of a passively mode locked ultrafast laser in order to create interesting pulse dynamics. Our inspiration for these experiments comes from Feedback Shift Registers which produce psudeo-random deterministic pulse sequences. Early successes in creating Analog Feedback Shift Registers and implementations of Optical Linear Feedback Shift Registers for slow pulses have guided this work [51].

In order explore the feedback mechanisms we simulate the essential cavity dynamics of a KLM laser as described in Chapter 3. Our simulations are seeded with a Gaussian pulse in the picosecond to 100 femtosecond range. The master equation governs the pulse evolution simulating one cavity round-trip of the pulse for each time step. Typical simulations converge on a steady state solution within 2,000 to 8,000 round trips. Our pulse widths are in the 30 to 50 fs range. The non-linear and dispersive effects were applied to the pulse using the Split-Step Fourier Transform

method [49].

Here we review feedback shift registers including LFSRs, AFSRs, and OLFSRs. We propose a passive KLM-LFSR cavity. We then review our analytical model for the laser system and discuss our simulator. Finally we present the results of our simulations and close with some new ideas about helpful representations.

## 4.1 Feedback Shift Registers

Linear Feedback Shift Registers (LFSR) are at the core of spread spectrum communication schemes such as CDMA. In a spread spectrum communication protocol transmitter sends a message by hopping to a series of frequencies in a pre-determined order. The receiver listens to the noisy channel on the same frequencies in the same order to capture and decode the message. The frequency hopping can be determined by a psuedo-random noise source such as an LFSR.

LFSRs produce a pseudo-random bit string via a simple shift register and an adder. A shift register of length $M$ produces $2^M - 1$ bit strings before repeating. The next input to the shift register is created using bits already in the register. This next bit is determined by

$$x_n = \sum_{i=1}^{M} a_i x_{n-i} \pmod 2,$$

where $x_n$ is the next bit in the sequence, and $M$ is the order of the LFSR. An LFSR [7] with $M = 4$ has values $a_{1-4} = (1, 0, 0, 1)$. This sequence will repeat after 15 bit strings are created.

Figure 4-1 shows a block diagram of an LFSR. The bit string looks random for lengths small compared to $2^{M-1}$.

### 4.1.1 Analog Feedback Shift Registers

An "Analog" Feedback Shift Register (AFSR) may have some advantages over a digital LFSR for spread spectrum communications [8, 52]. The LFSR phase acquisition period in digital communications can be quite slow. By approximating the acquisi-

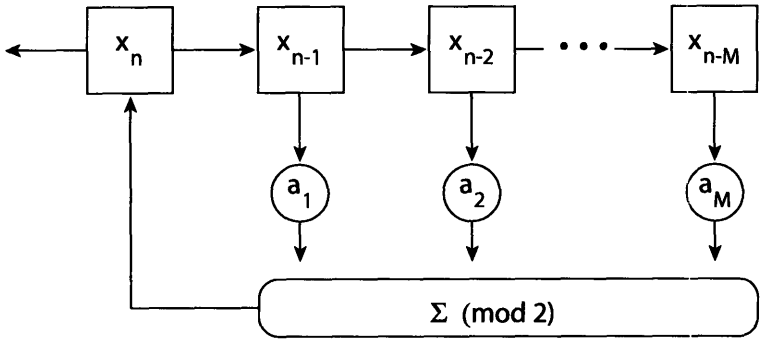



Figure 4-1: LFSR. Figure reproduced from [7]

tion process using analog values and a real valued map we can acquire more quickly at the expense of increased symbol error. Electronic hardware implementations of AFSRs have been proposed [50] for use in spread spectrum communications. Here we investigate a scheme for implementing a LFSR/AFSR-like psuedo-random pulse sequence of optical pulses in an ultrafast laser cavity.

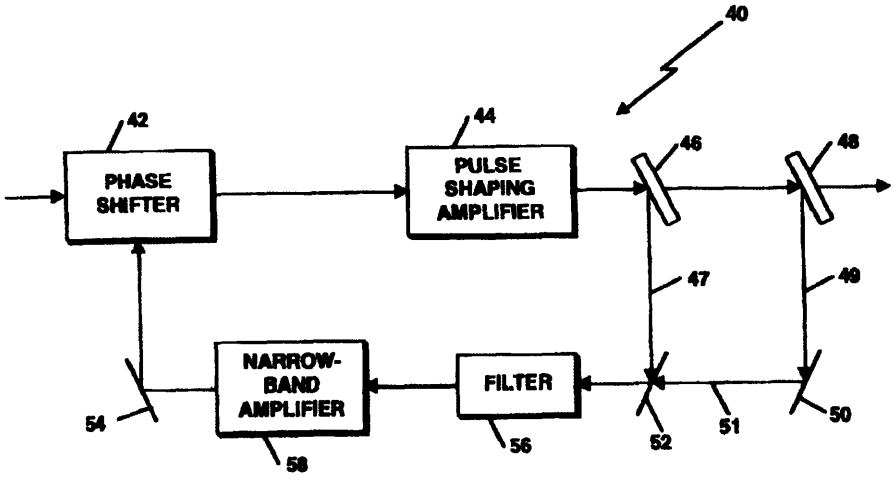



Figure 4-2: Optical AFSR implementation [8].

An optical AFSR is described in U.S. patent no. 5,729,388. A laser pulse train passes through a "phase shifter" and into a feedback loop (fig. 4-2). This phase shifter applies a phase shift to the pulses that is dependent on the intensity of the pulse from the feedback loop. If this phase shifter is calibrated to produce phase

shifts of 0 and $\pi$: a 0 phase when the feedback loop and phase shifter see two 0 phase pulses or two $\pi$ phase pulses and a $\pi$ phase shift for one 0 phase pulse and one $\pi$ phase pulse. The analog nature of the beam splitters and noise will produce pulses in between these extreme 0 and $\pi$ phase values. The filter, narrow-band amplifier, and phase shifter will need to implement this real-valued map:

$$x_n = \frac{1}{2}\left[1 - \cos\left(\pi \sum_{i=1}^{N} a_i x_{n-i}\right)\right]$$

Using sophisticated electronics to implement this system is near trivial. Unfortunately, electronics are limited by a scaling which is slow compared to modern optical communication. Fully optical versions of an LFSR have been proposed and patented.

### 4.1.2 Optical Linear Feedback Shift Registers

An implementation of a Optical Linear Feedback Shift Registered (OLFSR) is described in U.S. patent 5,208,705 [51]. The crucial piece in this implementation employees a Sagnac optical switch [53]. This optical switch is a non-linear optical device where a transmitted pulse propagating in a fiber cable acquires either a $\pi$ or $2\pi$ phase shift depending on the counter-propagating switching pump intensity.

This switching is made possible by Self Phase Modulation (SPM) brought on by the Kerr effect in the fiber. The index of refraction increases in the presence of an intense electric field such as a short laser pulse. This increase in the index of refraction leads to a change in phase of the pulse.

The non-linear index of refraction due to the Kerr effect is

$$\begin{aligned} n &= n_0 + n_2 I \\ &= n_0 + n_2 \frac{P}{A_{\text{eff}}} \end{aligned}$$

where

$$n_0 = 1.76,\ n_2 = 3 \text{ x } 10^{-20}\ m^2 W^{-1}$$

for Ti:Sapphire.

We can expect a phase shift due to SPM caused by the Kerr effect to be

$$\Delta\phi = -\frac{2\pi}{\lambda} L n_2 I(t), \tag{4.1}$$

where $\lambda$ is the wavelength, $L$ is the crystal length, and $I$ is the intensity $I = P/A_{\text{eff}}$. In Sagnac optical switches long lengths of fiber are used and the pulses are focused tightly to affect a suitably large phase shift.

### 4.1.3 Kerr Lens Mode-locked Linear Feedback Shift Registers

We propose a new implementation of an optical feedback shift register based on the mode-locking action of ultrafast lasers.

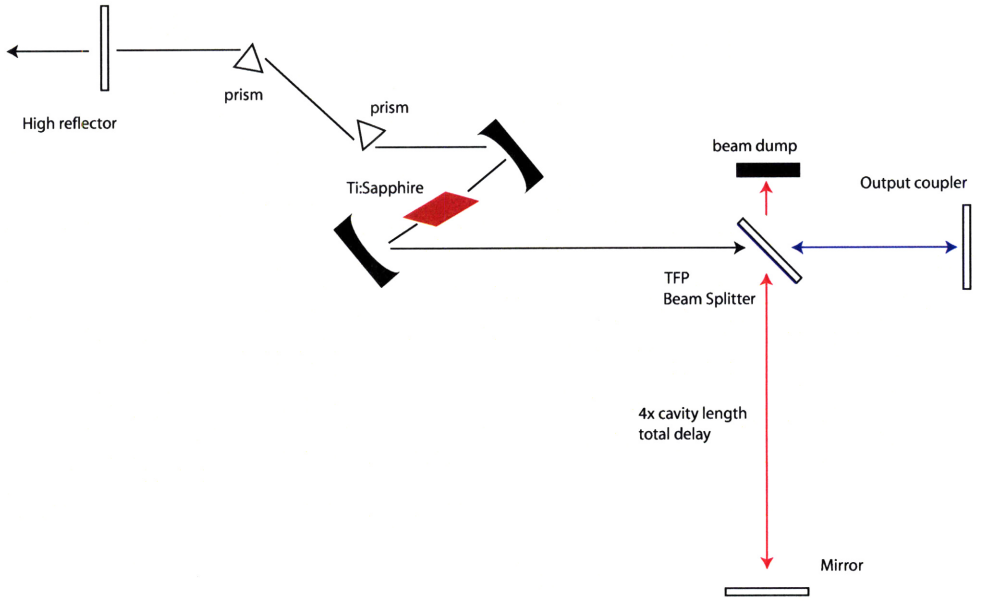



Figure 4-3: Proposed KLM AFSR

The dynamics of a Kerr-Lens Mode-Locked (KLM) laser cavity converge on a steady-state ultrashort pulse train that minimizes loss in the cavity. By changing the

cavity configuration it's possible to modify the loss to select for other sequences. In particular, we propose that by modulating the properties of the gain medium using the previous pulses out of the cavity, the coherent optical evolution can converge on the pseudo-random sequence of a Linear Feedback Shift Register (LFSR). Two such coupled cavities can be understood as solving a recursive state estimation problem, relating a mathematical program for the constrained optimization to the cavity dynamics.

A typical Ti:Sapphire oscillator produces 50fs pulses at 400mW ML output power, 90MHz repetition rate, and a 3% output coupler. This implies a 14W intra-cavity power or 1.55 x $10^{-7}$ J/pulse corresponding to a pulse intensity $I = E/\tau = 3.1$ MW. We assume effective area of the pulse to be $(25\mu\text{m})^2$ [54].

From equation (4.1) we can calculate the magnitude of a typical phase shift within a Ti:Sapphire crystal.

$$\begin{aligned}
\Delta\phi_{25\mu} &= -\frac{2\pi}{\lambda} L n_2 I(t) \\
&= -\frac{2\pi}{800\text{nm}} 1\text{mm } 3 \text{ x } 10^{-20}\ m^2 W^{-1}\ 3.1\ \text{MW}/(25\mu\text{m})^2 \\
&= -1.169
\end{aligned}$$

This phase shift is in the right range for a $0/\pi$ phase switching scheme as proposed for optical AFSRs. The number calculated above is for the full power of the pulse. The main problem with our proposed optical AFSR scheme isn't one of range of SPM, but the lack of a non-linear switching element in the cavity.

## 4.2 The Simulator

The simulator described in Chapter 3 evolves temporal and spectral dynamics. All spatial cavity modes are assumed to be $\text{TEM}_{00}$. It is necessary to seed the simulator with a short pulse. The gain of the cavity is negative for everything except a short pulse. This protects against the build-up of noise but makes the cavities not self-starting.

The split step Fourier transform method uses an iterative method to simulate the non-linear SAM and SPM action of the cavity. Generally this method converges within two iterations. Typically the iterations only fail to converge when an insufficient number of points are used for the envelope or when the model parameters cause a pulse to have excessive gain. For each round-trip of the laser cavity a minimum of 4 Fourier transforms are performed. Most simulations performed 30,000 Fourier transforms while converging on a pulse.

## 4.3 Simulations

All of our simulations are based on Figure 4-3. The precise cavity alignment depends on the method of feedback. We found no cavity configurations that resulted in periodic or chaotic pulse sequences.

One purely subtractive feedback method could be called "gain robbing." The TFP Beam Splitter is set to send a small percentage of the light down the long feedback arm. The end mirror of that arm is moved in slightly from the perfect 2x cavity length. The feedback pulse returns into the cavity, and arrives into the Ti:Sapphire crystal before the large intra-cavity pulse. This smaller feedback pulse is amplified by the gain medium and, because the alignment is a little off, then misses the first prism and hits a beam block. The primary pulse arrives at the gain medium before it has had a chance to recover its population inversion. The intra-cavity pulse is amplified, but not as much as is normal due to the gain robber feedback pulse.

Figure 4-4 shows the final pulse in the top window and the bandwidth of that pulse below it. The pulse width and amplitude are also shown as a function of round-trips which are single steps of the simulation. No modulation of any kind was detected in the width or amplitude of the pulse. Because gain robbing method is only subtractive, it is not a good approximation to the XOR function seen in an LFSR. The pulse width this system converged to was wider than the same laser cavity without feedback. This wider pulse would be consistent with a slightly larger cavity loss value. The pulse width was 38.6 fs.

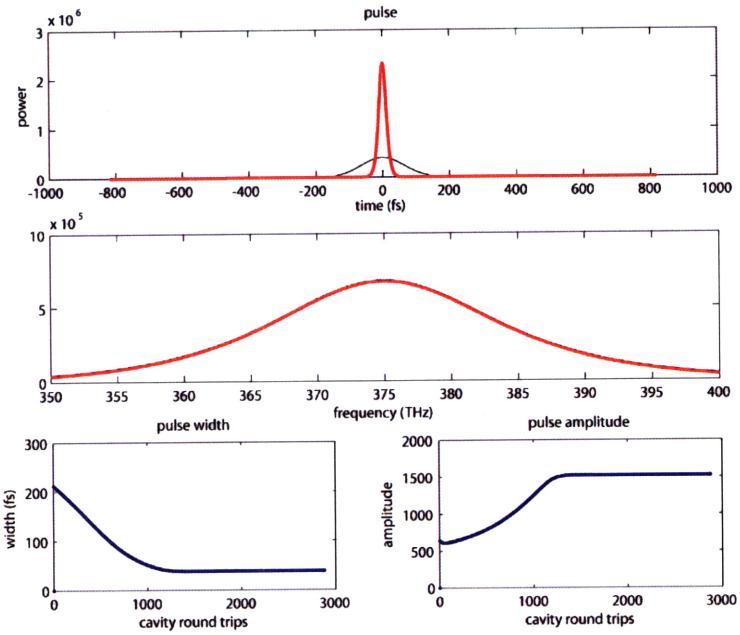



Figure 4-4: job43: "Gain Robbing" feedback into the Ti:Sapphire crystal before main cavity pulse arrives–No modulation.

After exhausting many variations on gain robbing we decided to use a 50%/50% beam splitter in place of the TFP shown (fig. 4-3). This system would be most analogous to the optical AFSR system. The modulation would have to be entirely due to loss in the cavity encouraging other modes to excite.

This 50/50 beam splitter experiment had the exact same pulse width and amplitude as an experiment where 100% of the pulse stayed inside the cavity. The only interesting part of the feedback was the coherent addition of the 1st and 4th pulses. While the last set of experiments seemed to be too subtractive, these seemed too additive. We varied the relative phases between the feedback arm and the cavity and found a similar effect seen earlier. The pulse width and amplitude changed corresponding to an increased loss in the cavity. We added dispersion to the feedback pulse that would correspond to going through lenses. This widened the pulse but produced no periodic behavior. By adding a time delay to the feedback arm which would be equivalent to the feedback arm being slightly longer than an integer multiple of the cavity length we got a frequency shift pulse train.

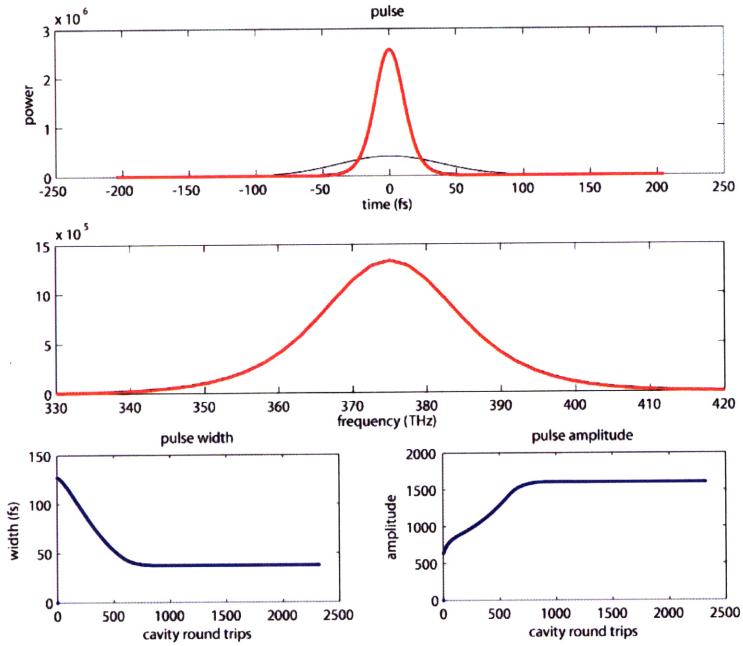



Figure 4-5: job66: 50%/50% Beam Splitter 1:4 LFSR

## 4.4 Discussion

The three optical elements consisting of the output coupler, the TFP, and the feedback mirror could be viewed as just one mirror with a frequency dependent reflection. This Mach-Zehnder filter has filter characteristics dependent on the relative difference of the cavity lengths of the two cavities.

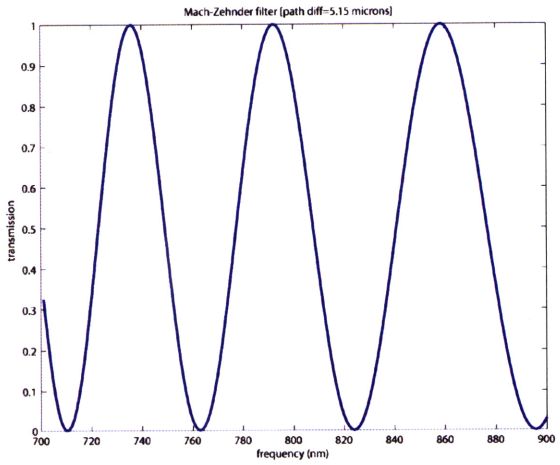



Figure 4-6: Mach-Zehnder filter path difference of $5\mu$.

Figures 4-6 and 4-7 show the transmission of the Mach-Zehnder filters as a function

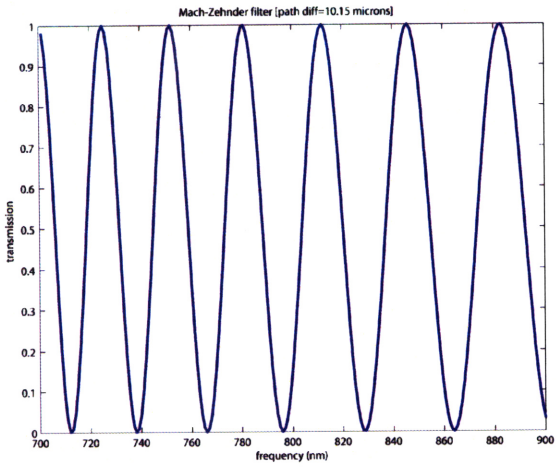



Figure 4-7: Mach-Zehnder filter path difference of $10\mu$.

of frequency. We have assumed that the feedback length is set to the length of the cavity. Using these filters should allow us to tune the cavity bandwidth by adjusting the feedback delay line.

The key feature we have overlooked in our examination of LFSRs is the (mod 2) XOR that is performed after the addition. This is the key operator that produces psuedo-random pulse sequences. The XOR is a non-linear operator. We will not be able to make a good approximation of it using linear optics. Mirrors, beam splitters, delays, simple losses, etc. will not approximate this function. An element similar to a Sagnac optical switch will be necessary.

We may want to consider using a bi-stable representation for our bits. These simulations assumed a modulation of analog amplitude or pulse width values. Three types of cavity configurations are possible that should produce a bi-stable system: space, time and frequency. Two spatial cavity modes could be simultaneously aligned. Pulse energy in one or the other mode could indicate a bit. A second system could be based on getting multiple pulses excited in the cavity simultaneously. An amplitude modulator could set the rough temporal spacing. With no further modifications only one pulse would ring up in the cavity. With the addition of a cavity element that would increase loss with increased pulse amplitude more gain would be freed up to excite other modes. A similar laser has been made for use in optical networking [55]. Finally, we could use two frequencies. We could install a spatial/spectral filter between the

prisms and the high reflector to shape the cavity loss as a function of frequency. This loss could force the laser to lase in one of two frequencies with a longer pulse width due to the limited bandwidth. Given these two stable frequencies a modulation scheme sending power from one frequency to another using a non-linear element in the feedback arm could be developed.

## 4.5 Conclusion

The KLM cavity dynamics do not provide an easy bit representation for information processing. The relatively low per-pass-gain ($\approx$ 1%) limits the possibility of fast bit switching. Under all simulated cavity feedback configurations, the KLM cavity converged on an uninteresting train of single pulses.

It may be better to avoid fighting the tendency of these laser cavities to converge on a single self-similar train of pulses. If we were able to modulate the relative phase of the pulses without disturbing their amplitude or pulse width, we may be able to implement a phase-shift keying bit encoding. One non-optical way of modulating the phase is by placing a piezo on an end mirror to modulate the effective cavity length. An all-optical method for effective cavity length modulation may prove to be an effective route to an AFSR-like implementation.

# Chapter 5

# "Finding" a Pulse Shape

In the previous chapter we discovered that pulses in a KLM cavity strongly converge to a steady stream of ultrafast pulses. Here we attempt to discover the function that the laser cavity minimizes.

## 5.1 Analytic Examination

"The laser cavity finds the pulse that minimizes loss–it's like magic." Here we will attempt to find the function that the laser minimizes while ringing-up. Ideally we would like to discover a Lyapunov-like function–a monotonically decreasing, positive definite function.

### 5.1.1 Complex Ginzburg-Landau Equation

The Haus master equation (3.7) describing the full KLM dynamics including soliton effects is a complex Ginzburg-Landau equation (CGLE). For each cavity round-trip the gain parameter $g$ is fixed. Gain depletion attenuates the gain parameter as the total energy in the pulse approaches the limit of the gain media $E_{sat}$.

$$g(T) = \frac{g_0}{1 + \frac{E_P(T)}{E_{sat}}}$$

where

$$E_P(T) = \int_{-\infty}^{+\infty} |A(T,t)|^2 dt$$

This changing parameter of our non-linear partial differential equation makes solving the full ring-up dynamics difficult. It is usually difficult to find an analytic solution to the CGLE for any particular set of parameters. To simplify this analysis we will consider only the dynamics of the laser near full pulse energy $\delta g \dashrightarrow 0$.

Our complex Ginzburg-Landau equation describing the laser dynamics around the stationary solution is:

$$\frac{\partial}{\partial T} A(T,t) = (g-l)A + \left(D_{gf} + jD\right)\frac{\partial^2}{\partial t^2}A + (\gamma - j\delta)|A|^2 A \tag{5.1}$$

This equation has a soliton-like solution [56]

$$a(t) = A_0 \ \mathrm{sech}^{(1+\mathrm{j}\beta)}\left(\frac{\mathrm{t}}{\tau}\right),$$

and a continuous wave solution

$$a(t) = A_0 \exp(-j\omega t).$$

The CGLE:

$$\frac{\partial}{\partial T} A = A + (1 + jc_1)\frac{\partial^2}{\partial t^2}A - (1 + jc_2)|A|^2 A \tag{5.2}$$

The competition between these two lasing modes and relative stability of each determines whether the laser is self-starting or not. Where self-starting implies an immediate ring-up to the soliton-like solution. Soto-Crespo et. al. have extensively studied the CGLE approximated around the soliton and the CW solutions [57, 58]. Their analysis used a numerical simulation similar to the one presented earlier. They found regions of the CGLE parameter space where the CW lasing mode was unstable and the soliton lasing mode was stable which suggests that the laser should be "self-starting" in that region.

An analytic Lyapunov functional for the CW lasing mode has been reviewed and

confirmed numerically [59]. Unfortunately this treatment of the CGLE did not consider soliton solutions. In the full parameter space the solitons are usually unstable and much of the space is chaotic. Rederiving the work in that paper to approximate soliton solutions is presently beyond the scope of this author.

We will attempt to further simplify the master equation.

### 5.1.2 Non-linear Schrodinger Equation

As we saw in the numerical section of this paper as the pulses get shorter the soliton effects begin to dominate. For ultrashort pulses we could consider the soliton effects as the primary dynamics and assume that gain, SAM, and filtering can be considered small pertibations.

$$\frac{\partial}{\partial T}A(T,t) = jD\frac{\partial^2}{\partial t^2}A - j\delta|A|^2A \tag{5.3}$$

Terms this equation is identical to solitons propagating down a fiber. It has no dissipative terms so the total energy in the pulse will remain constant. This equation is known as the Non-Linear Schrodinger Equation (NLSE).

We have now restricted ourselves to an area very near the stationary solution to the Haus master equation. The Lyapunov function for this equation is well known [60]. Adapted for our NLSE parameters we a Lyapunov function.

$$V = \int_{-\infty}^{+\infty} dt\left[-D|A|^2 + \frac{1}{2}\frac{\delta}{2}|A|^4a + |\frac{\partial}{\partial t}A|^2\right]$$

### 5.1.3 Numerical Confirmation

This Lyapunov function is nothing great. We found a fractal.

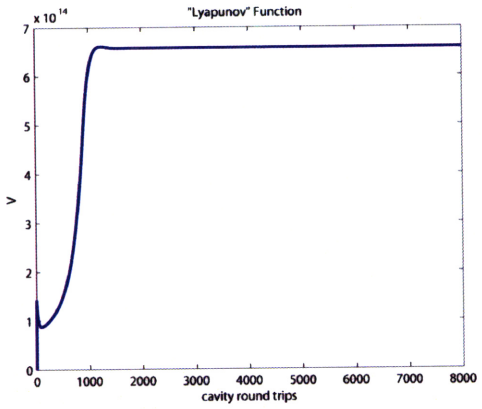



Figure 5-1: job89b_1: The Lyapunov function derived from the NLSE for the CGLE.

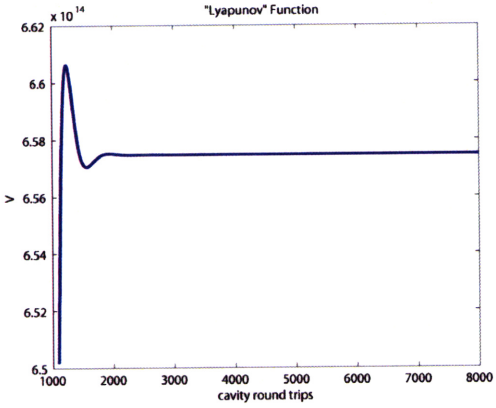



Figure 5-2: job89b_2: The Lyapunov function derived from the NLSE for the CGLE.

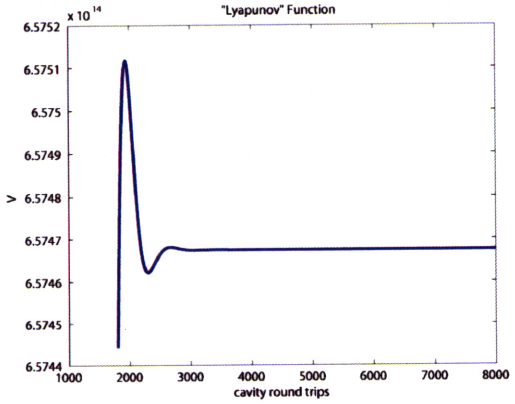



Figure 5-3: job89b_3: The Lyapunov function derived from the NLSE for the CGLE.

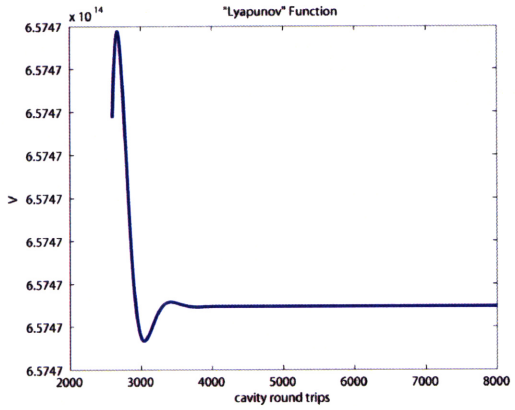



Figure 5-4: job89b_4: The Lyapunov function derived from the NLSE for the CGLE.

# Chapter 6

# Spatial Light Modulation with the Triplet State

The previous three chapters reviewed KLM cavity dynamics, inter-cavity bit representations and pulse conversion properties. We found the cavity properties insufficient for manipulating pulse trains. At this point, I realized that the molecules from my earlier electron triplet state work could add the necessary degrees of freedom to the laser cavity to modify pulses. Here we present a new all-optical spatial light modulator that will give us a "dial" into the laser cavity to manipulate the ultrafast pulses (Chapter 8).

We created a molecular triplet state spatial light modulator (SLM) to be used both outside and inside the laser cavity for ultrafast pulse shaping. The SLM consists of a liquid or thin film sample with a strong triplet state absorption. The molecule is selected to be transparent to the target light before pumping and strongly absorptive when pumped into the triplet state. The sample is exposed to laser light reflected off of a DMD chip to produce a 2D pattern to spatially populate the triplet ground state. When the target light is shown through the sample it is transmitted or absorbed according to the programmed pattern. This is, to our knowledge, the first all-optical ultrafast pulse shaper and the first all-optical inter-cavity spatial frequency modulator.

The spatial light modulator is used in prism based pulse shaper configuration.

Prism-based pulse shapers have far less loss [61] ($\approx 3\%$) compared to the more commonly used grating pulse shapers [18] ($\approx 70\%$). The prism configuration is already present in the design of most KLM cavities (Chapter 7).

## 6.1 Candidate Molecules

Our interest in using the triplet state of a molecule as a optical switch arose from our experience with ZnTPP and CuTPP in our quantum control experiments described in Chapter 2. As shown in Figure 6.1 the photo-induced triplet state absorption spectra is as high as 30% near 500nm for CuTPP pumped at 400nm. This sample was prepared to have a concentration $\approx$ 2 x $10^{-7}$M in toluene. This large triplet state absorption is due to the molecule's large ISC quantum efficiency (63% from Table 6.1) and the intense instantaneous pump power of 300$\mu$J per pulse corresponding to a power of 3 gigawatts of power over a $\approx$1mm$^2$ area. The power per cm$^2$ of our pump is extraordinarily high 30 terawatts per cm$^2$. Compared to a typical power for modulating a C60 spatial light modulator of 100 megawatts/cm$^2$ [62] our quantum control experiments had plenty of power.

Metalloporphyrins such as CuTPP and ZnTPP have recently been proposed for use in all-optical switching applications [63]. Our absorption measurements agree with the results of Singh, et. al. Unfortunately the triplet state absorption of metalloporphyrins is particularly low around 800nm (Figure 1-3). Table 6.1 lists the properties of other candidate molecules we've considered for use in a triplet absorption spatial light modulator for our pulse shaper.

ZnTPP, CuTPP and quaterthiophene, the molecules we used in our early quantum control experiments, have little or no absorption at 800nm where our laser cavity produces pulses. Two polymers, P3HT and P3OT, have far stronger absorptions at 800nm. P3HT and P3OT are both long chain oligothiophenes like the 4-mer quaterthiophene we were familiar with. Their strong singlet-singlet absorption at 532nm and triplet-triplet absorption in the 800nm range made them ideal candidates for use in our triplet SLM.

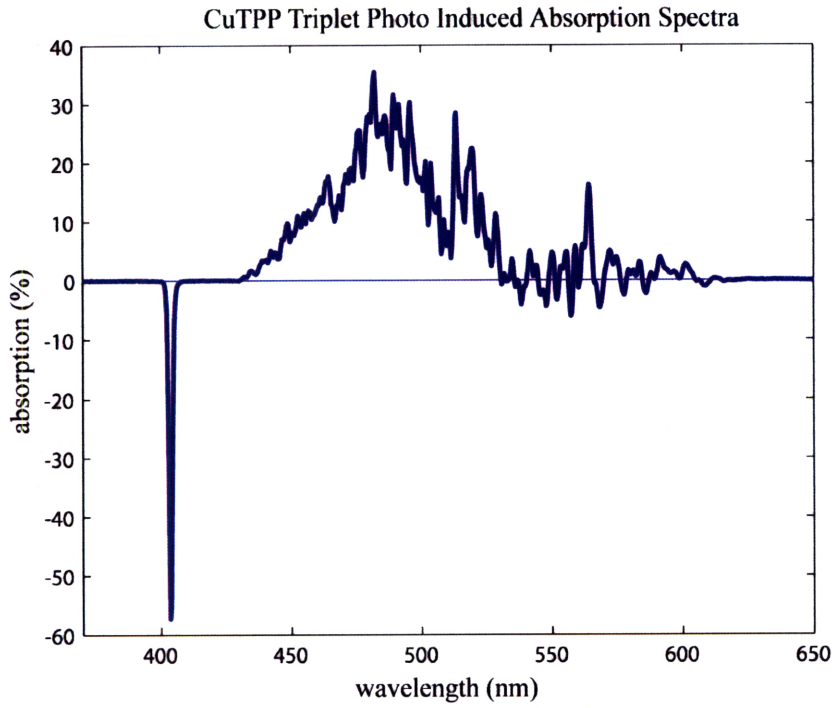



Figure 6-1: CuTPP PIA

Phenosafranine and Safranine are dyes with strong absorptions near 532nm and triplet-triplet absorptions at 800nm. We chose phenosafranine for our thin film experiments. The 280$\mu$s triplet lifetime time constant is a little lower than we might want for CW pumping.

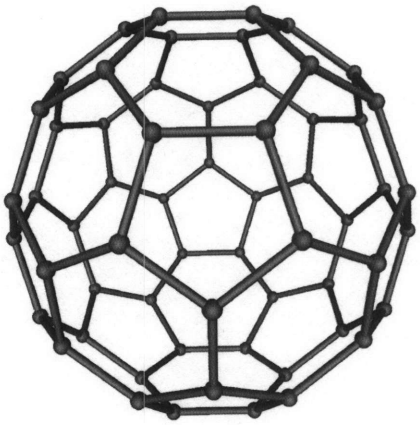
## 6.2 2D Molecular Spatial Light Modulator

We used a Digital Micro-Mirror Device (DMD) in an Infocus LP120 projector to spatially modulate our pump light. The resulting 2D pattern was projected onto a liquid sample in our pulse shaping apparatus (Figure 7-8). Commercial projectors have been used to create inexpensive photo-lithography systems by inverting the optics and focusing the light into a stereomicroscope [67]. We found the relatively weak output of the lamp to be insufficient for pumping our samples.

In order to attain high enough pumping energies we needed to remove the lamp, power supply and color wheel from the projector and replace it with a 532nm 2 watt

| name | singlet $\lambda$ | triplet $\lambda$ | ISC $\phi$ | $\tau_T$ |
|---|---|---|---|---|
| Zn tetraphenylporphyrin (ZnTPP) | $\approx$ 425nm a | $\approx$ 475nm a | 0.88 [16] | 2ms [16] |
| Cu tetraphenylporphyrin (CuTPP) | 425 a | 475 a | 0.67 [16] | 2ms e |
| quaterthiophene (4T) | 390 [64] | 475-525 a | | |
| poly-(3-octylthiophene) (P3OT) | 443 [65] | 825 [65] | | |
| poly-(3-hexylthiophene) (P3HT) | 455 [65] | 825 b | | |
| Buckminsterfullerene (C60) | | | | |
| phenosafranine (PS+) | 527 [66] | 800 [66] | 0.28 [66] | 280us |
| safranine (SF+) | 529 [66] | 822 [66] | 0.31 [66] | |

Table 6.1: Table of molecules. (a) measured, (b) assumed. Solvents used: CuTPP, ZnTPP, 4T and C60 in Toluene. P3HT and P3OT in Xylenes. Phenosafranine and Safranine in Methanol.



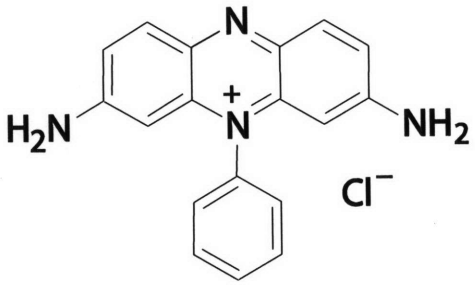



Figure 6-2: C60 (image by Michael Strock distributed under GNU Free Documentation License) and Phenosafranine

Coherent Verdi CW laser. Removing these components from the projector required a bit of hacking.

### 6.2.1 Projector Hacking

The Infocus LP 120 projector is one of the smallest projectors in the Infocus line of products. We chose this model at the suggestion of Rehmi Post who recently hacked the same projector for a One Laptop per Child prototype. This model projector was particularly accessible to hacking due to the specialized nature of its subsystems. The lamp power supply and lamp were connected to the rest of the projector by a single


Figure 6-3: Infocus LP 120 projector $1100 in 2006

data cable. The color wheel was driven by a three phase disk driver chip. The fan was unnecessary once the power supply was removed. The DMD uses a large heat sync for cooling.

Each of the three subsystems were monitored by the projector's processor. The data cable leading to the lamp power supply relayed lamp temperature information to the projector's processor so it could shutdown the lamp to avoid overheating. One ulterior motivation for the processor to closely monitor the lamp is to sell more Infocus brand lamp replacements. Their business model probably relies on the profits from replacement bulbs so they made it difficult to run the projector without one of their bulbs installed.

It turned out that the the protocol for communicating from the lamp to the projector was 4800 baud 8n1 RS232 serial. I made a cable, constructed a serial monitoring device from a couple of 888 op amps and use a null modem cable and a laptop to record projector boot up traffic. I then removed the lamp power supply, lamp and fan and emulated the boot traffic with a program written in C for my laptop. It would have been preferable to use microprocessor to emulate the lamp, but my desk wasn't setup for microprocessor programming at the time. The code to emulate the lamp is included in Appendix D. The color-wheel was replaced with three 15 Ohm resistors is a Y configuration.


Figure 6-4: open projector

### 6.2.2 Optics for spectrometer cells and thin films

We measured projector performance using two sets of optics. For the 1mm spectrometer cells we used the 20mm focusing lens that came with the projector followed by a -80mm lens. The light from the DMD is reflected off of a curved mirror of an unknown focal length. The 20mm/-80mm lens combination created a focused image about 45mm infront of the projector with a depth of focus of at least 2mm. Figure 6.2.2 shows the images projected into samples of P3HT in Xylenes. The transmission through the projector and lenses was 32%. This assembly produced pixels of 30 microns on each side.

For thin films we were able to used a Mitutoyo M Plan APO 10 microscope objective lens. The lens has a depth of focus of 3.5 microns. This lens had a small effective aperture, as a result the transmission efficiency through the projector for the pump beam was less than 10%. Figure 6.2.2 shows the projected image recorded by a CMOS imager placed in the focal plane. This lens produced pixels of $8\mu m$ x $6\mu m$.

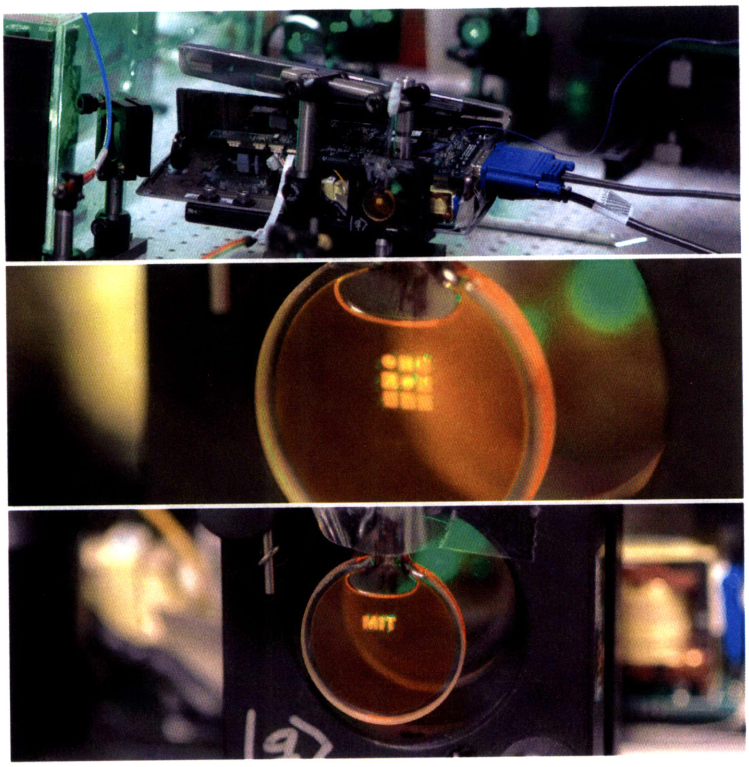



Figure 6-5: The projector pumping patterns onto P3HT: the Center for Bits and Atoms logo and MIT. The images shown here are at a magnification corresponding to $30\mu m$ per pixel.

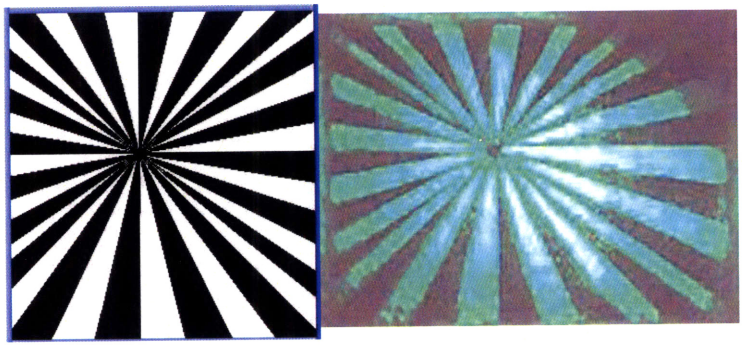


Figure 6-6: Image using a Mitutoyo M Plan APO 10 Objective lens. The image on the is 2.27mm x 1.74mm corresponding to a pixel size of $8\mu m$ x $6\mu m$.

# Chapter 7

# Ultrafast Pulse Shaping using Molecules

Here I describe my all-optical triplet state pulse shaper. I developed this pulse shaper as a step to my goal of building an intra-cavity pulse shaper. By using a triplet molecule in solution as our spatial light modulator I attained far less loss than the AOM pulse shaper we used in our earlier experiments. It is important to note, however, that unlike the AOM pulse shaper, this is an amplitude only pulse shaper that has no ability to affect the phase of the pulse.

## 7.1 Prism-Based Pulse Shaper

The prism-based pulse shaper design was chosen for its low loss ($\approx$ 3%) and how it closely resembles the negative dispersion prism configuration in our KLM cavity. A recent article published by Lioudakis, et. al., demonstrates a low loss prism-based pulse shaper [61]. A review article from 2000 covers most current pulse shaper designs [18].

We placed our molecular sample in the Fourier plane of the stretcher-compressor prism pairs as shown in Figure 7-1. The index of refraction of the prisms is a function of the wavelength of light. The ultrafast pulses typically had a bandwidth of 30nm centered at 800nm. The first prism spatially distributed the light as a function of

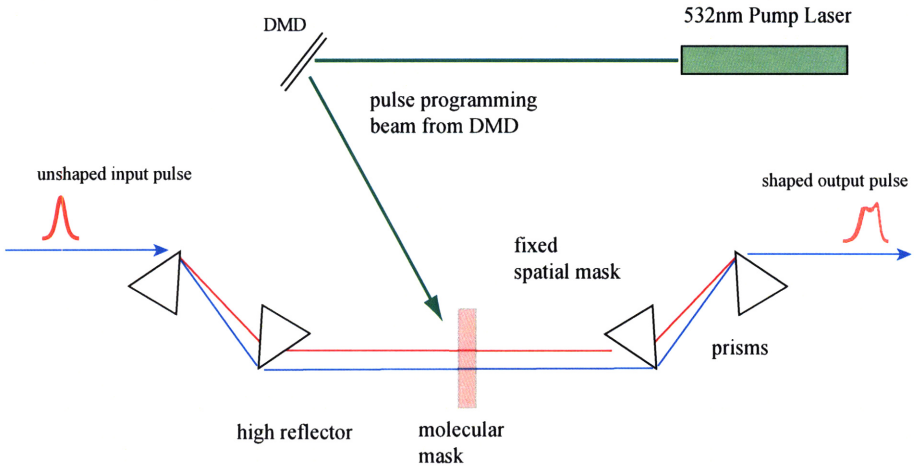



Figure 7-1: Pulse Shaper

frequency and the second prism collimated the frequency components. The width of the frequency distribution was somewhere around 2 to 3.5 mm. The light then passed through the molecular sample and was transformed back into the time domain by the second set of prisms. Thin films, PDMS doped substrates and liquid samples were used as targets for the programmed light.

## 7.2 Frequency Resolved Optical Gating

The shaped pulses were characterized using a model 8-20 GRENOUILLE from Swamp Optics. The GRENOUILLE is a descendant of a Frequency Resolved Optical Gating (FROG) device [68]. The output of a FROG is a 2D plot of frequency verses time. A polarization-gate FROG operates by splitting the pulse to be measured into two beams and delaying one a time $\tau$ with respect to the other. The undelayed beam passes through a polarizer, a piece of nonlinear (Kerr) material, another polarizer rotated 90 degrees with respect to the other, and finally into a spectrometer. The delayed beam is focused to intersect with the undelayed beam in the Kerr material. The intensity of the delayed beam causes a polarization rotation in the undelayed beam allowing it to pass through the second polarizer and into the spectrometer.

$$E_{\text{sig}}(t,\tau) = E(t)|E(t-\tau)|^2$$

The spectrogram incident on the spectrometer is given by:

$$I_{\mathrm{FROG}}^{\mathrm{PG}} = |\int_{\infty}^{-\infty} E(t)|E(t-\tau)|^2 \exp(i\omega t)dt|^2 \tag{7.1}$$

As the delayed pulse is swept through the undelayed pulse many spectrograms are recorded. The intensity profile of the pulse modulates the amplitude of the spectrum off the pulse. With a little reflection you can convince yourself that all of the phase and time information is recorded in the set of spectrograms. By performing a two-dimensional phase-retrieval the original $E(t)$ can be recovered.

The FROG software (VideoFROG by MesaPhotonics) solves the phase-retrieval problem in real time and displays a continually updating frequency vs time plot of the recorded pulse.

## 7.3 Genetic Algorithm

In order to control the pulse shaper I wrote a program which implemented a genetic algorithm to search for optimal pulse shapes. The program displayed vertical bars (shown in Figure 7-2) on a second video card where the output was split and sent to a LCD and the hacked projector to control the DMD. The pattern shown on the screen was the pattern projected on the sample cell.

The GA retrieved pulses by a memory mapped interface to the VideoFROG software. The operation of the Genetic Algorithm is described in Appendix B.

From the complex frequency trace generated by the VideoFROG software we plot the Husimi distribution [27] of time and frequency. Given a complex frequency domain representation of the pulse $E(\nu)$ the Husimi is calculated from:

$$Q(t,\nu) = \int\int dt'd\nu' S(t',v')e^{-(\nu-\nu')^2-(t-t')^2} \tag{7.2}$$

where $S(t,\nu)$ is the Wigner distribution:

$$S(t,\nu) = \int E(\nu+\nu')E*(\nu-\nu')e^{2i\nu' t}d\nu' \tag{7.3}$$

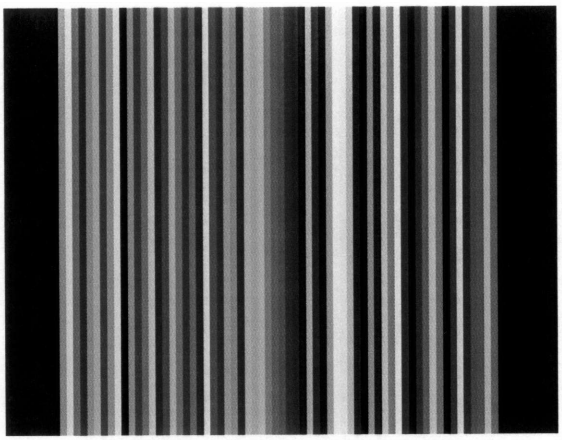


Figure 7-2: The GA generated patterns like this which were projected onto the molecular sample to shape the pulse.

## 7.4 Sample Preparation

We used three forms of molecular masks: a liquid in a spectrophotometer cell, a thick PDMS film doped with the molecule and a mirror or microscope slide with the molecule deposited bare on the surface. The liquid samples were prepared using fresh dry solvents and degassed in several freeze thaw cycles.

The PDMS film shown in Figure 7-3 allowed for a strong singlet-singlet absorption. PDMS doped with a laser dye has been used as an optical filter [69]. Unfortunately for our application PDMS is far too permeable to air. Oxygen would quickly quench our triplet state.

The thin films were prepared in a spin coater at 3000 RPM spun for approximately 90 seconds. The molecules, P3HT and phenosafranine, were deposited on to glass slides and dielectric mirrors. The P3HT is a long chain polymer and was relatively easy to spin coat. The regioregular P3HT was ordered from Aldrich and used without purification. The average molecular weight of the polymer was 17500. The P3HT samples were prepared in a 39.3mg in 100ml of p-Xylenes, also ordered from Aldrich. Phenosafranine was ordered from Aldrich and dissolved in anhydrous methanol. The phenosafranine was prepared in a high concentration of 1.2 x $10^{-2}$ M. A 50ml volume was thoroughly degassed for use in liquid and thin film molecular masks. The thin films were prepared at these concentrations using the spin coater. Typically 20 to

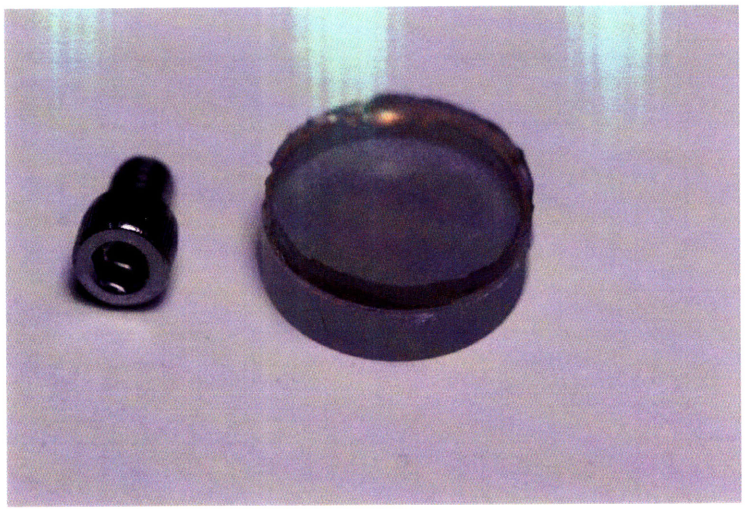


Figure 7-3: Thick PDMS layer with C60 on a dielectric mirror.

30 drops at a rate of 1 drop per second resulted in optical densities near 0.2. The phenosafranine on the dielectric mirror difficult to prepare. The 800nm dielectric mirrors were prepared by Quality Thin Films using alternating layers of hafnium oxide and silicon oxide. Typically about one out of three attempts to coat the mirrors with phenosafranine worked. The phenosafranine dissolves easily in methanol and isn't at all viscous both qualities made the spin coating process result in mirrors that were sparsely or barely coated at all. Working phenosafranine coated mirrors had an optical density of 0.19 at 532nm. About 0.05 of that was reflected light from the dielectric coating. Thin film samples were left under a flow of nitrogen to remove oxygen.

## 7.5 Results

This pulse shaper worked, but it worked for the wrong reason. The genetic algorithm was able to consistently find pulse shapes to maximize fitness functions. However, it was almost definitely not using the triplet state. The sample heated at relatively low (150mW) pump powers and caused spatial distortions in the shaped pulse.

Our pulse shaper design counted on the fact that a 2W CW laser would be able to pump enough population into the triplet ground state to attain a sufficient triplet-

triplet optical density to attenuate portions of the pulse spectrum. When the light is focused down to a small area and the triplet state is long ($\approx$ 1-2ms) then the triplet ground state can be populated relatively easily. In the case of phenosafranine with a triplet lifetime of 280$\mu$s and a 2D projected image this proved to be a problem.

Using a typical pump power of 850mW we would have 2.28 x $10^{18}$ photons incident on the DMD. With an efficiency of 68% only 1.55 x $10^{18}$ could arrive at the sample. Each bar (as displayed in Figure 7-2) might have 1/30th of that power which brings us to 5.16 x $10^{16}$. This number of photons will be available over one second to pump the sample into the triplet state. In the case of a moderately short lived triplet state like phenosafranine (280$\mu$s), we will end up with a steady triplet state population in the neighborhood of 2.8 x $10^{13}$. Longer lived triplet state molecules such as ZnTPP and CuTPP might have triplet state lifetimes in the milliseconds which would provide for another factor of 10 towards more population in the triplet ground state. To get an idea of the triplet optical density it is best to compare it with the molecular density. For a high liquid sample concentration of 1.45 x $10^{-3}$ we will have approximately 4.4 x $10^{15}$ molecules in a 1mm square volume through the sample. This isn't terribly far off for a longer lived triplet state.

In the following sections show pulse shaping. Unfortunately it is very unlikely that the triplet state was involved in generating the pulse shapes. We observed heavy spatial mode distortion at higher powers indicating we were heating our sample. The recovery time between displaying a pulse shape and displaying a blank screen was too long to be the triplet state of our molecule.

The pulse shapers did work consistently though.

### 7.5.1 Minimize Pulse Width

In this section we show the resulting pulse shapes from the pulse shaper using a C60/toluene solution. We instructed the GA to find the pulse shape that minimized the pulse width.

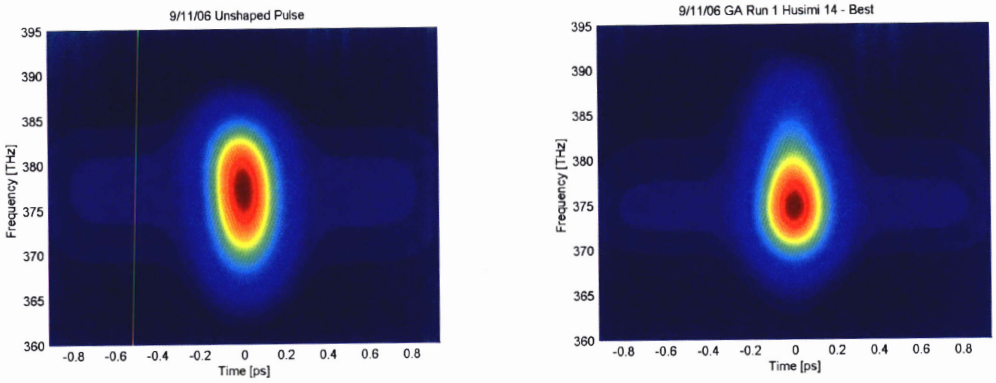



Figure 7-4: GA run to minimize pulse width. Husimi Plots of unshaped (left) and shaped (pulses)

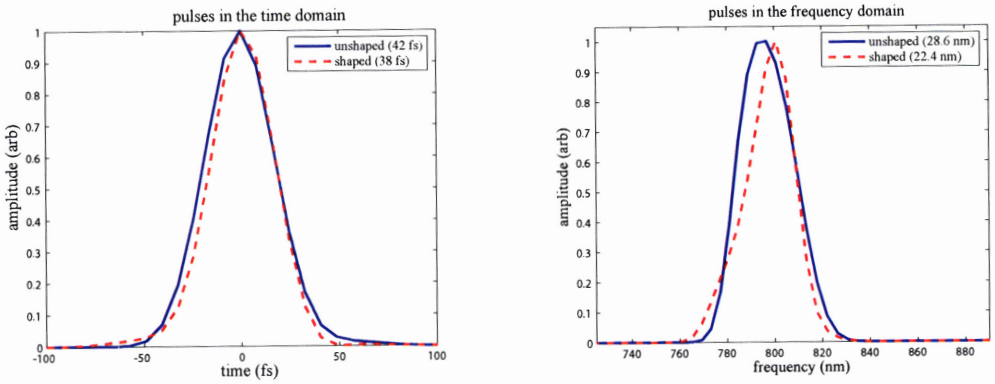



Figure 7-5: GA run to minimize pulse width. Time and Frequency domain plots

### 7.5.2 Time Domain Double Pulses

In this section we gave the GA an arbitrary goal to match. In Figure 7.5.2 the black shortest pulsed line shows the goal for the GA. Considering I was overheating the sample, it did a decent job of finding working pulse shapes.

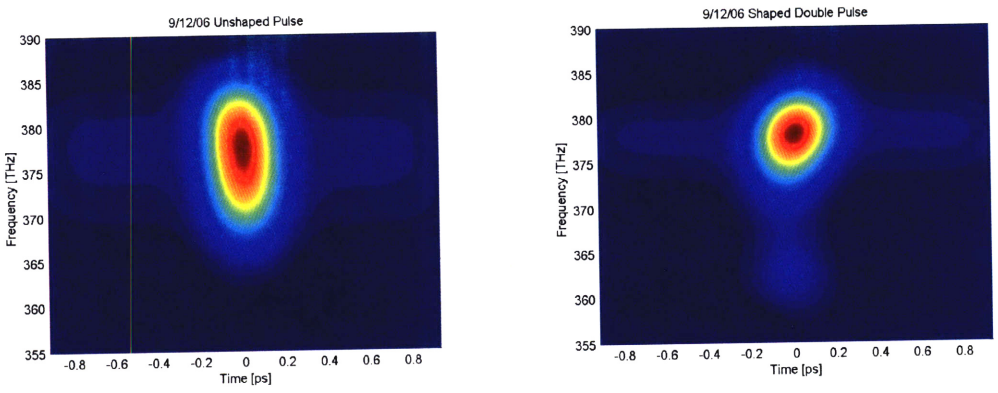



Figure 7-6: GA run to fit a double pulse goal. Husimi Plots of unshaped (left) and shaped (right)

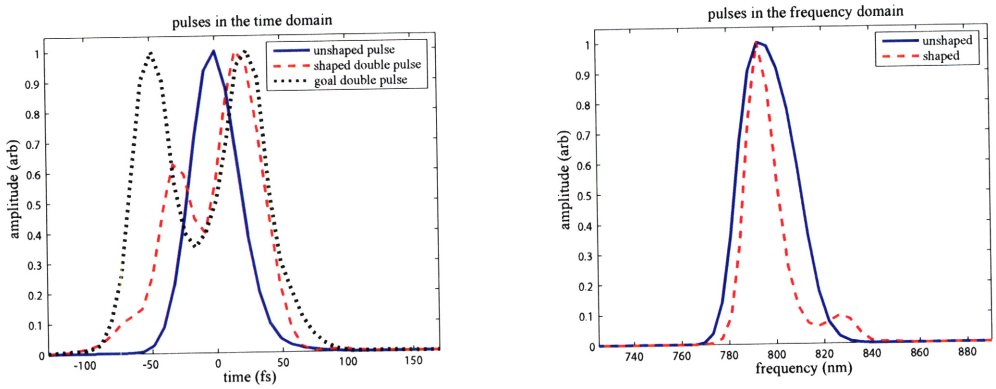



Figure 7-7: GA run to fit a double pulse goal. Time and Frequency domain plots

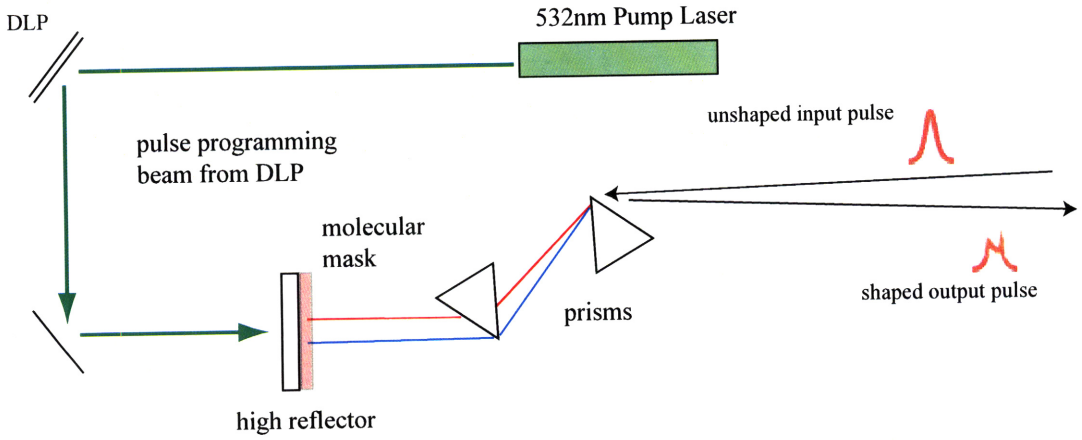



Figure 7-8: Molecular Mask Pulse Shaper

# Chapter 8

# Intra-Cavity Pulse Shaper

Here we present our all-optical intra-cavity pulse shaper. Because of the lasers's bi-stability and relative fragility of mode-locked operation, we had no ambitions of making a pulse shaper to create arbitrary pulses. Instead, we were interested in how we can introduce a modulator inside the cavity and work with the laser's gain and soliton properties to produce pulse trains. Ultimately, we would like to go further to close the loop between our quantum control experiments and our intra-cavity pulse shaping to produce a feedback system where the search for an optimal pulse is discovered through the evolution of the laser cavity with feedback. Our intra-cavity pulse shaper is a first step in creating a route to modifying the cavity based on an external stimulus.

## 8.1 Experimental Setup

Thin films of phenosafranine were prepared on dielectric mirrors as described in Section 7.4. The high reflector mirror after the prism pair was removed from our KM Labs Ti:Sapphire and replaced with a phenosafranine coated mirror as shown in Figure 8-1. The laser was returned to mode-locked operation and showed no degradation in performance due to the thin film. The lasers was allowed to run for two hours mode-locked to check for thin film damage caused by the high intra-cavity pulse power–none was found.

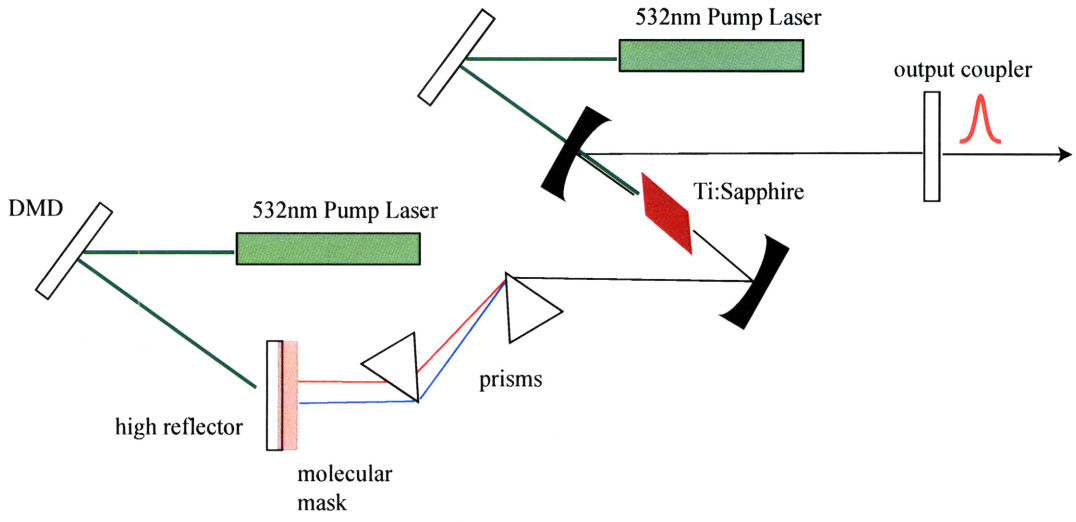



Figure 8-1: Intra-Cavity Molecular Mask Pulse Shaper

Because the external-cavity pulse shaper from the previous chapter lacked sufficient instantaneous pump power to fill the triplet ground state, I decided to focus the CW beam on to a spot size roughly comparable to the spot of the intra-cavity pulse on the covered mirror. The smaller spot size increased the photon flux incident on the thin film which allowed for a larger population in the triplet state. To minimize the effect of oxygen quenching of the triplet state I flowed nitrogen over the thin film for several hours and kept the laser cavity under positive nitrogen pressure.

## 8.2 Perturbing the Cavity Loss

The initial alignment was performed at high power (800mW) with a spot size under $1mm^2$. When the green pump caused the laser to drop out of mode-locked operation I decreased the power and attempted to affect the shape of the pulse without loosing mode-locked operation.

Figures 8-2 and 8-3 shows the Ti:Sapphire laser output with a 600mW pump incident on a small portion of the thin film. The shaped Husimi plot on the right of Figure 8-2 indicates an increase in pulse bandwidth. The explanation is slightly

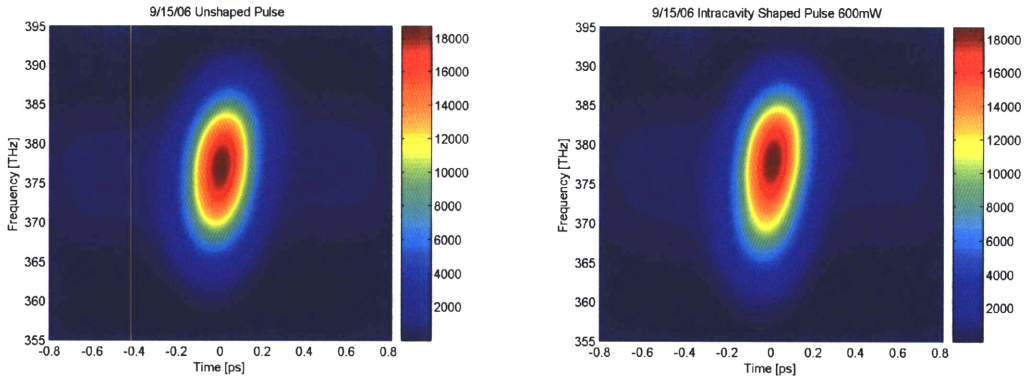



Figure 8-2: Husimi Plots of unshaped (left) and shaped (right) using a phenosafranine thin film. 060915c4,c5

counterintuitive. The triplet-triplet absorption of the molecule is increasing the loss of the cavity at a particular frequency. One might expect the attenuation of one frequency component to result in a narrower frequency distribution. As we learned from our earlier investigation in Chapters 4 and 5 the laser does not "find the pulse shape to minimize loss," but instead lases in any way possible. In this case, the Husimi distribution (the laser pulse inside the cavity) has shifted towards higher frequencies.

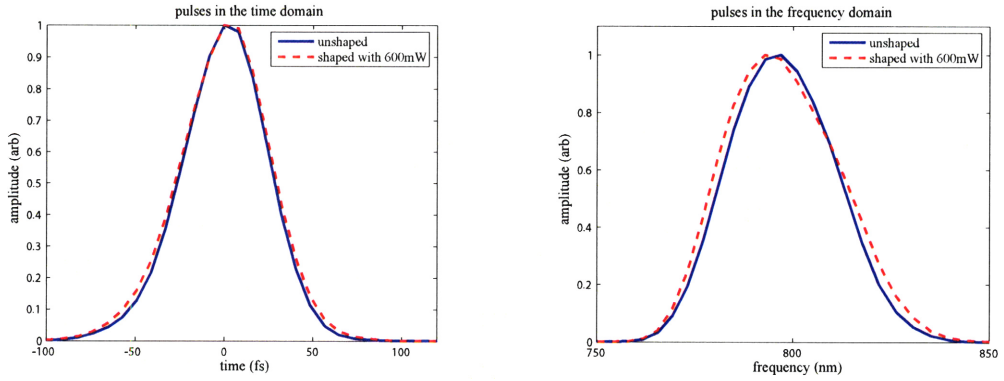



Figure 8-3: Time and Frequency domain plots of shaped and unshaped pulses using a phenosafranine thin film.

This shift is also shown in the plot of the absolute value of the frequency components. The magnitude of the shift is $\approx$ 4nm. The pump beam was focused with a 250mm lens. I varied the spot size of the pump on the film and eventually damaged the film at a power of 1.2W and a the focus only a few millimeters from the thin film.

The laser remained able to mode-lock the sample was just blown off of the surface.

I replicated these experiments again after rotating the mirror to an undamaged portion of the thin film and returning the laser to mode-locked operation. Again, I found that I was able to shift the center frequency of the pulse this time at a lower power. Figures 8-4 and 8-5 show the similar results. Pump powers as low as 100mW had a noticeable affect on the FROG spectra, but the difference was difficult to distinguish in two static images.

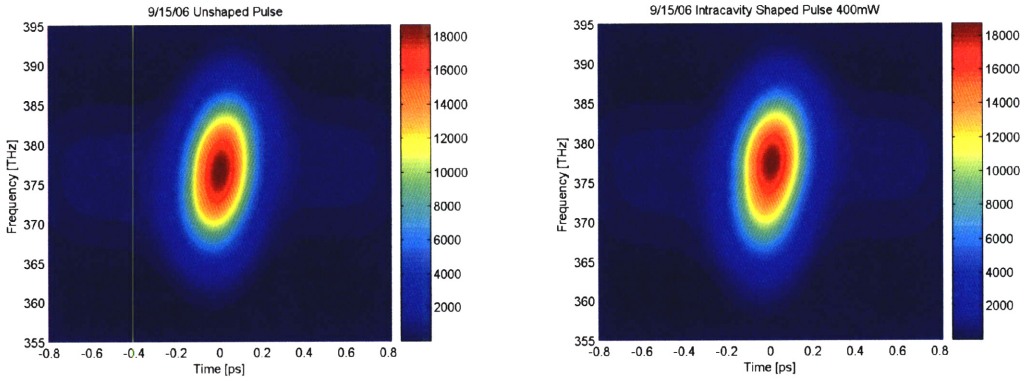



Figure 8-4: Husimi Plots of unshaped (left) and shaped (right) using a phenosafranine thin film. 060915c6,c7

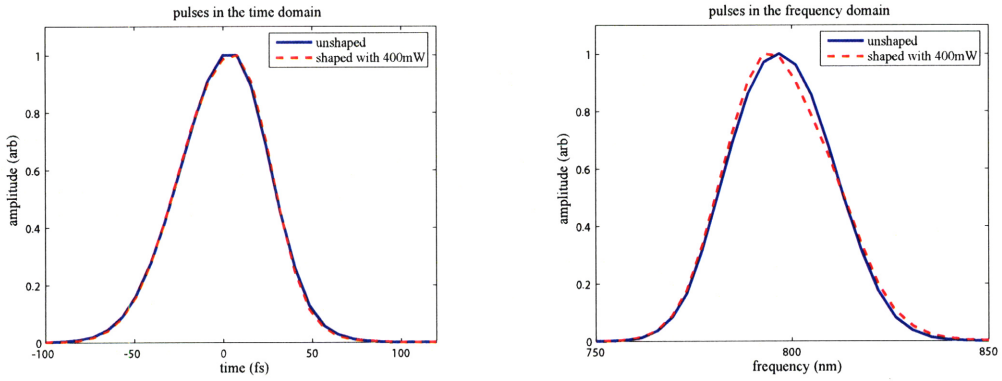



Figure 8-5: Time and Frequency domain plots of shaped and unshaped pulses using a phenosafranine thin film.

## 8.3 Richer Dynamics

Using the CW pump to excite the thin film allows for only a limited set of pulse shapes to be observed. The pulses bouncing in the cavity have converged on a single self-similar train of pulses. As we saw in Chapter 4 it is reasonably difficult to get the laser cavity to converge on anything other than a self-similar train of pulses.

Adding a time component to the pumping of the light modulator and gating the output pulses reveals pulses in transition similar to those seen in the ring up of Figure 3-13. Figure 8-6 shows the evolution of a shaped pulse in the laser cavity. Using a q-switched pump will allow for a temporarily higher optical density without increasing the heating of the sample. Proper synchronization between the q-switched pump and the Ti:Sapphire pulse train will reduce the sensitivity of the system to oxygen. A short-lived triplet state is irrelevant if the time between pumping the sample and the inter-cavity light is very small.

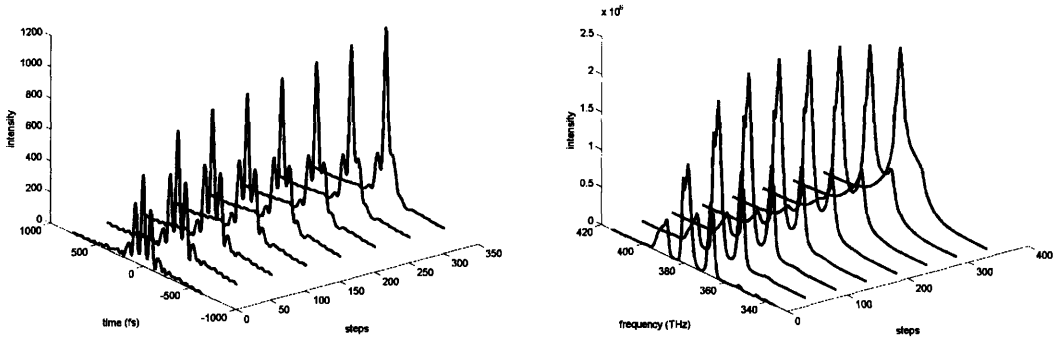



Figure 8-6: Time and Frequency domain plots of the evolution of a shaped pulse in a laser cavity with no mask. This is equivalent to a triplet pulse shaper with a triplet lifetime that is short compared to the round trip time of the cavity (a typical round trip time is 10ns).

# Chapter 9

# Results and Discussion

In my early experiments we used shaped ultrafast pulses to maximize the triplet state of molecules. This experiment demonstrated that the shaped light was better suited to controlling the absorption of light into the singlet state than perturbing rates of inter-system crossing. This triplet experiment lead to our interest in ultrafast laser cavities and their ability to process information. These cavities, left relatively unmodified, tend to converge on a train of self-similar ultrashort pulses. In my cavity simulations, representing bits in the form of pulse amplitude or pulse width did not produce a periodic modulation of the pulse train. Finally, I decided to combine elements from the first two experiments to produce two new devices: an extra-cavity pulse shaper that uses the triplet state of molecules as a spatial light modulator to shape pulses in the frequency domain, and an intra-cavity pulse shaper which selectively attenuates the frequency components of an intra-cavity pulse.

In short, I sought to use lasers to control molecules and found that I was able to use molecules to control lasers.

## 9.1 Contributions

I created new pulse shaper which, with the right pump, will use the triplet states of molecules to selectively attenuate frequency components of an ultrafast pulse. This extra-cavity pulse shaper was a step towards creating an intra-cavity pulse shaper to

control the frequency components of a cavity.

My intra-cavity pulse shaper optically attenuates frequency components in a Kerr-Lens Mode-Locked cavity. This new control into a KLM cavity has several possible consequences. It may lead to feedback experiments where the dynamics of a KLM cavity solve an information processing problem such as an Analog Feedback Shift Register or implement a message passing algorithm. It could allow for useful optical tuning of the pulse shape to compensate for extra-cavity shaping effects. Most ambitiously, this new cavity element opens up the possibility of performing a feedback experiment where the light from an extra cavity experiment is feedback into the laser cavity to modify the frequency components of successive pulses. The ring-up of the laser cavity under these conditions could perform an all-optical quantum control experiment.

## 9.2 Extra-Cavity Pulse Shaper Improvements

Heat dissipation is a large problem for this pulse shaper in CW mode. It can be solved with two modifications: a flow cell and a nanosecond pump. Flowing the sample alone would be counterproductive in terms of triplet state lifetime in front of the target pulse. Dyes in dye lasers are flowed, in part, to minimize the long-lived triplet states of dyes. Because we want to use this triplet state, we will have to pump the sample with a large number of photos in a short time to establish sufficient triplet optical density. With the right timing of pump pulse and target pulse this will be an effective amplitude pulse shaper. A q-switched pump with a power density of $100MW/cm^2$ might be a good start.

A considerable amount of work could be done on optics for the projector. Simply increasing the numerical aperture of the optics would increase the depth of field which would allow for a higher resolution pulse shaper.

## 9.3 Intra-Cavity Pulse Shaper Improvements

Much work can be done on selecting a better combination of triplet molecule and substrate. Several polymers are better candidates than PDMS including P3HT and possibly a P3HT/C60 combination. Ideally a thin substrate doped with triplet molecule and sealed from oxygen would allow for the longest lived triplet states. Alternatively, as we showed in Section 8.3 a shorter lived triplet state may be preferable.

A q-switched pump synchronized to the rep-rate of the Ti:Sapphire oscillator would be very useful in controlling pulse shapes inside the cavity. Adding a time component to the modulation of the triplet state would allow for greater access to the cavity dynamics. This pulse shaper could be used in conjunction with a pulse picker to select a single interesting pulse from the series of pulses in the cavity.

## 9.4 Future Directions

These triplet spatial light modulators attenuate spectral components of ultrafast pulses. It seems reasonable that they could be replaced with a gain material to amplify individual frequency components. Changing the spatial light modulator to a material to modify phase as a function of pumping would make an ideal phase pulse shaper. Oligothiophene crystalline films have several interesting properties that may be useful in this direction [70].

Further work on information processing in the KLM cavities should investigate a phase-shift keying bit representation. Using the amplitude modulator developed here, a double pulse bit representation may now be useful.

Ultimately we envision using this new access into the laser cavity to remove the computer from the quantum control experiment. By feeding the residual light from an extra cavity experiment back into the laser cavity it may be able to perform an all-optical search for efficient pulse shapes.

## 9.5 Acknowledgments

This work was supported by the FOCUS Center, NSF PHY-0114336 and the Center for Bits and Atoms, NSF CCR-1022419 as well as through NSF grants 9987916 and 0078972.

# Appendix A

# KLM simulator code

klmrun single.m - This code sets up the variables necessary for the simulation and then calls klmss.m 8000 times execute each round-trip time in the cavity.

```
clear all
close all

dt = 1e-16; % time step (=0.1 fs)
nt = 2^14; % Kalashnikov uses 2^13
T = nt*dt; % FFT window size (s)
P0 = 500000; % peak power
tp0 = 150e-15; % pulse width 1 ps
D = -0.275/1e15^2; % -75 fs^2
D = -75e-30; % -75 fs^2
D = -1.1815e-30;
D = -0.75e-30;
delta = 3.84e-8/10;  % 400mW 50fs D=75fs^2
delta = 9.1469e-10*6;   % 400mW 50fs D=75fs^2
l = 0.025; % loss
g0 = 0.07; % initial gain 0.07
%g_Esat = 1.56e-8;
g_Esat = 5.56e-8;
q0 = 0.01; % 0.005
gamma = 5e-9;
Omega_g = 2*pi*43e12; % 2*pi*43 THz
Omega_f = 270e12;
Omega_f = 970e12;
D_gf = 1/Omega_g^2;
P_A = 0;
```

```
Tau_A = 0; % 50 fs

t = ((1:nt)'-(nt+1)/2)*dt;              % vector of t values (s)
w = 2*pi*[(0:nt/2-1),(-nt/2:-1)]'/T;    % vector of w values (rad/ps)
fp = (2*pi*[(-nt/2:nt/2-1)]'/T + 2*pi*3e8/800e-9) / 1e12 / (2*pi);  % vector of w for

E_A = 18e-9; % 17 nJ


tic

figure
set(gcf,'DoubleBuffer','On');

track_width = 0;
track_height = 0;

a0 = gaussian(t,0,tp0,P0);

track_a0 = a0;
last_a0 = a0;
num_a0_same = 0;
track_fft_a0 = abs(fftshift(fft(a0)));

maxiter = 10;
tol = 1e-5;

a_start = a0;

for kk = 1:7999,
a1 = klmss(a0,t,g0,g_Esat,l,q0,P_A,D,delta,E_A,
            Tau_A,Omega_g,w,maxiter,tol,D_gf,dt,Omega_f,gamma);
the_max = max(abs(a1));
the_bins = sum(abs(a1)>(the_max/2))*dt;
track_height = [track_height;the_max];
track_width = [track_width;the_bins];

 if mod(kk,100)==0,
 track_a0 = [track_a0,a0];
 track_fft_a0 = [track_fft_a0,abs(fftshift(fft(a1)))];
 if (norm(last_a0-a0,2)/norm(a0,2) < 0.1)
num_a0_same = num_a0_same + 1
 else
num_a0_same = 0;
  last_a0 = a0;
```

```
 end
 if (num_a0_same==10)
 break;
  end
 end

 if mod(kk,20)==0,
 subplot(3,1,1)
 plot(t.*1e15,a_start.^2,'k',t.*1e15,abs(a1).^2,'r')
 ylabel('power');
 xlabel('time (fs)');
 title('pulse');
 subplot(3,1,2)
 plot(fp,abs(fftshift(fft(a1))),'r')
 xlabel('frequency (THz)');
 v = axis;
 axis([350,400,v(3),v(4)]);
 subplot(3,2,5)
 plot(track_width.*1e15)
 title('pulse width')
 xlabel('cavity round trips');
 ylabel('width (fs)');
 subplot(3,2,6)
 plot(track_height)
 title('pulse amplitude')
 xlabel('cavity round trips');
 ylabel('amplitude');

 kk
 the_bins_fs = the_bins * 1e15
 the_max
 gamma_time_max2 = gamma*the_max^2

 drawnow
 end

a0 = a1;

end

toc
```

klmss.m

```
function a1 = klmss(a0,t,g0,g_Esat,l,q0,P_A,D,delta,
            E_A,Tau_A,Omega_g,w,maxiter,tol,D_gf,dt,Omega_f,gamma)
%KLMSS Perform one round trip of a KLM oscillator
%   A1 = KLMSS(A0,DT,MAXITER,TOL) numerically computes one round trip of
%   a KLM oscillator. A0 is the starting waveform (typically gaussian), A1
%   is the resulting waveform. T is the vector of time values. MAXITER
%   is the maximum number of iterations in the split-step fourier method.
%   TOL is the tolerence for the split-step fourier method.
%
% This function performs one round trip of a KLM oscillator based on
% Haus master equation:
%
% Herman A. Haus. Mode-locking of lasers. IEEE Journal on Selected
% Topics in Quantum Electronics, 6(6):1173, 2000
%
% The simulation is based on work by Kartner et al:
% F.X. Kartner, Juerg Aus der Au, and U. Keller. Mode-locking with
% slow and fast saturable absorbers-what's the difference? IEEE Journal
% on Selected Topics in Quantum Electronics, 4(2):159-168, March 1998.
%
% Split-step fourier method based on:
% Agrawal, Govind.  Nonlinear Fiber Optics, 2nd ed.  Academic
% Press, 1995, Chapter 2
%
% Specific MATLAB implementation of the split-step fourier method
% is derived from the GPL'ed SSPROP package by Thomas E. Murphy
% (tem@alum.mit.edu).
%
% %%%%%%%%%%
%
% Copyright 2004, Jason M. Taylor
%
%

if exist('maxiter','var')==0
maxiter = 10;
end

if exist('tol','var')==0
tol = 1e-5;
end

% calculate gain with depletion
E_P = sum(abs(a0).^2) * dt;
```

```
g = g0 / (1 + E_P/g_Esat);

% calculate gamma the saturation coefficient
% gamma = q0/P_A;
% plot(gamma*abs(a0).^2)
% pause

% Calculate D_gf coefficient (ignore f for right now)
D_gf = g/Omega_g^2 + 1/Omega_f^2;

% START standard split-step fourier-transform method

halfstep = g - 1 - q0;
halfstep = halfstep - j*D*(w).^2;
halfstep = halfstep - D_gf*(w).^2;
halfstep = exp(halfstep/2);

a1 = a0;
afft = fft(a0);

uhalf = ifft(halfstep.*afft);

% iterate to find non-linear part

for ii = 1:maxiter,
   pulse_power = (abs(a1).^2 + abs(a0).^2)/2;
   midstep = min(q0*ones(size(pulse_power)),gamma*pulse_power);
   midstep = midstep -j*delta*pulse_power;
   uv = uhalf .* exp(midstep);
   afft = halfstep.*fft(uv);
   uv = ifft(afft);
   if (norm(uv-a1,2)/norm(a1,2) < tol)
a1 = uv;
break;
   else
a1 = uv;
   end
end

if (ii==maxiter)
   warning(sprintf('Failed to converge to %f in %d iterations',...
tol,maxiter));
   pause
end
```

# Appendix B

# Genetic Algorithms

Genetic Algorithms are commonly used to search rugged landscapes. They operate by manipulating a large number of variables to look for a solution.

Genetic algorithms (GAs) are computer search algorithms modeled after evolution. Natural evolution has arrived at a number of interesting solutions to maximizing the survivability of a species. GAs take a similar sensibility and apply it to solutions to a problem in a computer algorithm–the algorithm keeps the solutions that work and kills off the rest. The set of solutions that the computer tries could be said to evolve over time and hopefully converge on a best solution. This evolution is done by duplicating relatively successful solutions and mutating them. The unsuccessful solutions are thrown out.

Forrest [30] has a good review paper in Science on GAs. Mathematical modeling books are also good sources for information on GAs [7].

A search algorithm tries to maximize a fitness function. One could of course try every combination of inputs to a function and fully map out the the output space, but, given finite time, this is usually impossible.

GAs are good algorithms to use when the space is fairly flat but has a few regions of great interest. Shaping light to pump an electronic system is well suited for a GA. Most of the shapes will result in no major change in the absorption or emission spectrum but a few pulse shapes can accomplish what Gaussian pulses cannot.

## B.1 Algorithm Setup

GAs borrow most of their jargon from biology. The GA is said to *evolve* over time as it converges on a solution. In each *generation* or set of inputs to try, the GA tests each set of inputs, called *genes*, against the fitness function. The fitness function is maximized to find a good solution to the problem. The search is done by *mutating* genes and engaging in gene *crossover* where parts of two genes are combined to arrive at a new set of genes.

A basic implementation of a GA might have 100 genes which consist of strings of 8 bits. These bits could be interpreted as a decimal number and then used as an input to a fitness function. The fitness function could be a mathematical expression or, in our case, an experiment.

The way genes are interpreted is important when considering how to engage in the mutation and crossover steps. During a mutation, parts of the genes are changed randomly. If they are not changed in a reasonable way relative to the bit interpretation then the GA might have convergence problems. For example, if our GA is trying to find the decimal number 255 using an 8 bit binary gene then, if we have a gene with a value of 247, a random flipping of one of the bits will result in one of 246, 245, 243, 255, 231, 215, 183, or 119. For the case of searching for 255, it might be better to do the mutation in base 10 by adding a random number from -10 to 10. This mutation would be better suited to solving the problem.

A gene crossover step has similar hazards. Parameters from two genes can be combined to produce a new gene. It may make sense to group parameters together and call them a *chromosome*. Then when performing a crossover chromosomes may be kept intact.

## B.2 Basic Algorithm

Once a way to represent the genes is determined, programming of the GA may begin. First, allocate a set of genes and initialize them to random values. While the GA is

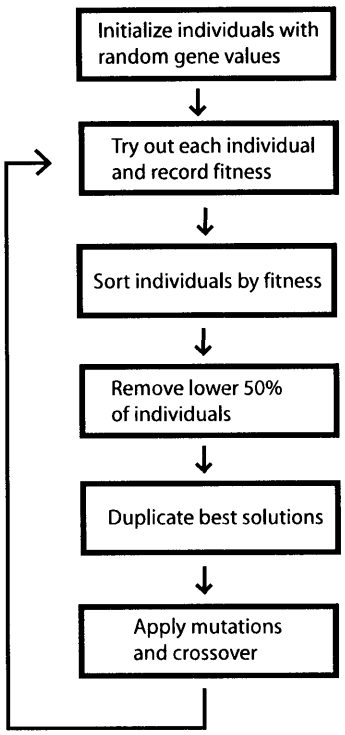



Figure B-1: GA Flow

running, it iterates in a loop. Figure B.2 shows the GA flow. First, apply each gene to the fitness function and record the fitness. Sort by fitness then drop bottom 50% or so. Duplicate the best solutions to fill the positions just dropped. Apply mutations and or crossover best suited to the problem. Then repeat.

The details of the implementation of this algorithm is strongly dependent on the problem. A bit of insight into how the the system behaves will generally help convergence. For example, knowing that a particular parameter will have either an exponential or polynomial effect on the resulting fitness, it may be useful to vary that parameter more slowly than another parameter with possibly a lower order effect.

The GA retains a large amount of information from previous generations when creating the next generation. Because of this, the GA should eventually have a set of genes where a few have a good fitness. This may take 20 to 100 generations depending on the type of problem and the parameters used.

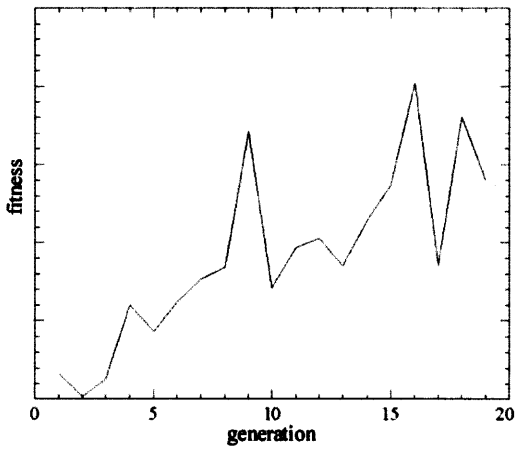



Figure B-2: This figure shows the progress of a GA running over many generations. The jagged behavior of the fitness indicates a lack of convergence.

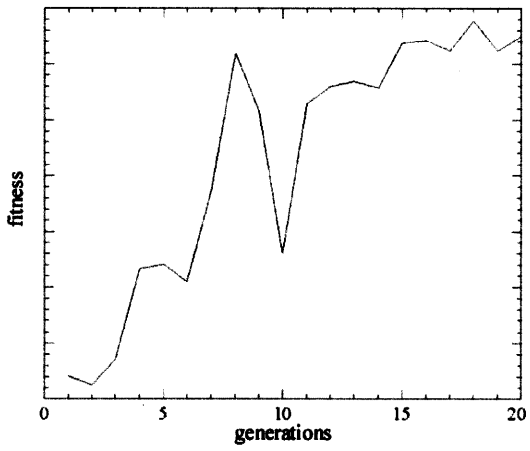



Figure B-3: This GA seems to have converged on a solution. The smooth behavior over the last few generations is a good indicator of convergence.

When a GA's solutions aren't getting much better as the generations iterate, then the algorithm should be stopped and the results examined. An analytic function that monitors the standard deviation could be used to determine when to halt the GA. It is also likely to be sufficient to just halt the GA after several generations and look at the fitness of the best genes as a function of generation. Figure B.2 shows the fitness as a function of generation for a GA that didn't converge. Figure B.2 is for a GA that did converge.

# Appendix C

# Genetic Algorithm Code

Here I include the code for the genetic algorithm I wrote for use in Chapter 7. The code presented here has been heavily edited to reduce the page count yet preserve demonstration of the GA and VideoFrog interface code. The GA routines are presented first, followed by the interface to the VideoFrog software which extracts pulse shapes from the Swamp Optics Grenouille 8-20, and finally the main body of the program.

## C.1 Genetic Algorithm Routines

### C.1.1 ga.cpp

```
#include "ga.h"
#include "videofrog.h"
#include <stdlib.h>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <direct.h>
#include <string.h>
#include "main.h"
#include <time.h>                                                    10
#include "ConStream.h"

struct individual GA_individuals[POPULATION];
struct individual next_gen_GA_individuals[POPULATION];
```

```
struct frog_data GA_frogdata[POPULATION];
double *goal;
unsigned int roulette[MAX_ROULETTE];
int GA_time_center=0;

int gacomp(const void *v1, const void *v2)
{
    struct individual *i1, *i2;

    i1 = (struct individual *)v1;
    i2 = (struct individual *)v2;

    return ((int)((i2->fitness - i1->fitness)*1000));
}

void rand_bars_for_individual(int ind_num)
{
    int i;

    for(i=0;i<MAX_GENES;i++)
        GA_individuals[ind_num].bars[i] = (rand()*255)/RAND_MAX;
}

void init_ga_vars(void)
{
    int i;
    GA_gen_num = 1;
    memset(GA_individuals,0,POPULATION*sizeof(struct individual));
    for(i=0;i<POPULATION;i++)
        rand_bars_for_individual(i);
    memset(GA_frogdata,0,POPULATION*sizeof(struct frog_data));

    // goal bars for testing GA performance
    /* memset(goal_bars,10,MAX_GENES);
    for(i=0;i<10;i++)
        goal_bars[i+15] = 200; */
}

void present_individual(int i)
{
    static unsigned char *screenptr=0;
    static char buf[512];
    char *pp;

    if (!screenptr) screenptr = (unsigned char*)malloc(BMP_IMG_SIZE);
```

```
    sprintf_s(buf,512,"presenting individual %d\n",i);
    // OutConStr(buf);
    pp = makeBarsImage(screenptr,GA_PRESENT_START,
        GA_PRESENT_END,GA_individuals[i].bars,MAX_GENES);
    memcpy(GA_individuals[i].pattern,pp,1024);
    GA_individuals[i].frog_index = i;
    GA_individuals[i].fitness = 0;
    LoadImageData(screenptr);
}

void collect_data_for_individual(int i)
{
RetrievePulse(&(GA_frogdata[GA_individuals[i].frog_index]));
}

void GA_find_time_center(void)
{
    unsigned int i;
    double max=0;
    int maxpos=0;
    char buf[512];
    static struct frog_data fd;

    RetrievePulse(&fd);

    for(i=0;i<frog_trace_size;i++)
        if (max<fd.abs_td[i]) {
            max = fd.abs_td[i];
            maxpos = i;
            sprintf_s(buf,512,"new max %f at position %d\n",
                max,maxpos);
            OutConStr(buf);
        }
    GA_time_center = maxpos;
}

double FITNESS_near_goal(double *abs_td)
{
    int i;
    double fitness=0;

    if (!goal) {
        OutConStr("goal not allocated!\n");
        return -1;
```

```
    }
    for(i=0;i<frog_trace_size;i++)
        // fitness += (double)1 - fabs(goal[i]-abs_td[i]);
        fitness += pow((double)1 - fabs(goal[i]-abs_td[i]),2);

    return fitness;                                                    110
}

double FITNESS_pulse_width(double PW, double power_meter)
{
    double fitness;

    fitness = PW;

    if (fitness>10)
        fitness = ((double)2000-fitness*10)*(power_meter);             120
    else
        fitness = 0;

    return fitness;
}

double FITNESS_double_pulse(double *abs_td) // old
{
    double max1=0, max2=0;
    int maxpos1=0, maxpos2=0;                                          130
    unsigned int i;
    double fitness=-20;

    if (GA_time_center > 0 && GA_time_center < frog_trace_size-1) {
        for(i=0;i<GA_time_center;i++)
            if (max1<abs_td[i]) {
                max1 = abs_td[i];
                maxpos1 = i;
            }
                                                                       140
        for(i=GA_time_center;i<frog_trace_size;i++)
            if (max2<abs_td[i]) {
                max2 = abs_td[i];
                maxpos2 = i;
            }

        fitness = max1 + max2 - (abs_td[GA_time_center])*2;
    }
```

```
    return fitness;                                                    150
}

void calculate_fitness_for_individual(int i)
{
    double fitness;
    struct frog_data *fdp;

    fdp = &(GA_frogdata[GA_individuals[i].frog_index]);

    //fitness = FITNESS_pulse_width(fdp->PW,fdp->power_meter);          160
    //fitness = FITNESS_double_pulse(fdp->abs_td);
    //fitness = FITNESS_max_peaks(fdp->abs_td);

    fitness = FITNESS_near_goal(fdp->abs_td);

    GA_individuals[i].fitness = fitness;
}

void make_roulette_index(void)
{                                                                      170
    int i,j;
    double total=0;
    double share;
    int rpos=0;
    char buf[512];

    memset(roulette,0,sizeof(int)*MAX_ROULETTE);
    for(i=0;i<POPULATION;i++)
        total += GA_individuals[i].fitness -
            GA_individuals[POPULATION-1].fitness;                      180
    for(i=0;i<POPULATION;i++) {
        share = ((GA_individuals[i].fitness -
            GA_individuals[POPULATION-1].fitness)*
            MAX_ROULETTE/total);
        for(j=rpos;j<share;j++)
            roulette[j] = i;
        rpos += (int)share;
    sprintf_s(buf,512,"%d fit %f share %d rpos %d\n",i,
            GA_individuals[i].fitness,share,rpos);
    OutConStr(buf);                                                    190
    }
}

void sort_generation_by_fitness(void)
```

```
{
    qsort(GA_individuals,POPULATION,sizeof(struct individual),gacomp);
}

void ga_present_individual(int i)
{                                                                    200
    present_individual(i);
}

void ga_eval_individual(int i)
{
    collect_data_for_individual(i);
    calculate_fitness_for_individual(i);
}

void ga_eval_population(void)                                        210
{
    int i;
    static char buf[512];

    sort_generation_by_fitness();
    make_roulette_index();

    sprintf_s(buf,512,"Individual Fitnesses\n");
    OutConStr(buf);
                                                                     220
    for(i=POPULATION-1;i>=0;i--) {
        sprintf_s(buf,512,
        "pop num: %d fitness: %f PW: %f frog index: %d\n",i,
        GA_individuals[i].fitness,
        GA_frogdata[GA_individuals[i].frog_index].PW,
        GA_individuals[i].frog_index);
        OutConStr(buf);
    }
}
                                                                     230
void two_point_crossover(struct individual *child1,struct individual *child2,
        struct individual *partner_A, struct individual *partner_B)
{
    int i;
    int ra, rb;

    ra = ((rand()*MAX_GENES)/RAND_MAX)%MAX_GENES;
    rb = ((rand()*MAX_GENES)/RAND_MAX)%MAX_GENES;
    if (ra>rb) {
```

```
        i = ra;                                                    240
        ra = rb;
        rb = i;
    }

    memset(child1,0,sizeof(struct individual));
    memset(child2,0,sizeof(struct individual));

    for(i=0;i<MAX_GENES;i++) {
        child1->bars[i] = (i>=ra&&i<=rb) ?
            partner_A->bars[i] : partner_B->bars[i];               250
        child2->bars[i] = (i>=ra&&i<=rb) ?
            partner_B->bars[i] : partner_A->bars[i];
    }
}

void average_crossover(struct individual *child1,
            struct individual *partner_A,
            struct individual *partner_B)
{
    int i;                                                         260

    memset(child1,0,sizeof(struct individual));
    for(i=0;i<MAX_GENES;i++) {
        child1->bars[i] = partner_A->bars[i]/2 +
            partner_B->bars[i]/2;
    }
}


void mutation(struct individual *child1, struct individual *parent)  270
{
    int i;
    int ra, rb;

    ra = ((rand()*MAX_GENES)/RAND_MAX)%MAX_GENES;
    rb = ((rand()*MAX_GENES)/RAND_MAX)%MAX_GENES;
    if (ra>rb) {
        i = ra;
        ra = rb;
        rb = i;                                                    280
    }

    memset(child1,0,sizeof(struct individual));
```

```
    for(i=0;i<MAX_GENES;i++) {
        child1->bars[i] = (i>=ra&&i<=rb) ?
            (rand()*255)/RAND_MAX : parent->bars[i];
    }
}
                                                                    290
void creep(struct individual *child1, struct individual *parent)
{
    int i;
    int ra, rb;

    ra = ((rand()*MAX_GENES)/RAND_MAX)%MAX_GENES;
    rb = ((rand()*MAX_GENES)/RAND_MAX)%MAX_GENES;
    if (ra>rb) {
        i = ra;
        ra = rb;                                                    300
        rb = i;
    }

    memset(child1,0,sizeof(struct individual));

    for(i=0;i<MAX_GENES;i++) {
        child1->bars[i] = (i>=ra&&i<=rb) ?
        parent->bars[i] + (rand()*26)/RAND_MAX - 13 :
        parent->bars[i];
    }                                                               310
}

void smoothing(struct individual *child1, struct individual *parent)
{
    int i;

    memset(child1,0,sizeof(struct individual));

    child1->bars[0] = parent->bars[0]/2+parent->bars[1]/2;
    for(i=1;i<MAX_GENES-1;i++) {                                    320
        child1->bars[i] = (unsigned char)(((int)parent->bars[i-1]
            + (int)parent->bars[i] + (int)parent->bars[i+1])/3);
    }
    child1->bars[MAX_GENES-1] =
    parent->bars[MAX_GENES-1]/2+parent->bars[MAX_GENES-2]/2;
}


void ga_evolve_population(void)
```

```
{                                                                    330
    int i=0;
    static char buf[512];
    int ra, rb;

    memset(next_gen_GA_individuals,0,
        POPULATION*sizeof(struct individual));

    for(i=0;i<5;i++) // keep best 5
        memcpy(next_gen_GA_individuals[i].bars,
        GA_individuals[i].bars,MAX_GENES);                           340

    for(i=5;i<POPULATION;i++) {
        ra = (rand()*MAX_ROULETTE)/RAND_MAX;
        rb = (rand()*MAX_ROULETTE)/RAND_MAX;
    switch(i%6) {
        case 0: // two point crossover skip one
        if (i+1 < POPULATION)
        two_point_crossover(&(next_gen_GA_individuals[i]),
            &(next_gen_GA_individuals[i+1]),
            &(GA_individuals[roulette[ra]]),                         350
            &(GA_individuals[roulette[rb]]));
        else
        memcpy(next_gen_GA_individuals[i].bars,
            GA_individuals[roulette[ra]].bars,MAX_GENES);
        i++;
        break;
        case 2: // average crossover
        average_crossover(&(next_gen_GA_individuals[i]),
            &(GA_individuals[roulette[ra]]),
            &(GA_individuals[roulette[rb]]));                        360
        break;
        case 3: // mutation
        mutation(&(next_gen_GA_individuals[i]),
            &(GA_individuals[roulette[ra]]));
        break;
        case 4: // creep
        creep(&(next_gen_GA_individuals[i]),
            &(GA_individuals[roulette[ra]]));
        break;
        case 5: // smoothing                                         370
        smoothing(&(next_gen_GA_individuals[i]),
            &(GA_individuals[roulette[ra]]));
        break;
```

```
        case 6: // keep best operators
        memcpy(next_gen_GA_individuals[i].bars,
            GA_individuals[roulette[ra]].bars,MAX_GENES);
        break;

    };                                                          380
    }
    memcpy(GA_individuals,next_gen_GA_individuals,
        POPULATION*sizeof(struct individual));

}
```

### C.1.2 ga.h

```
#ifndef _GA_H
#define _GA_H

#include "videofrog.h"

#define MAX_GENES 64
#define POPULATION 60

struct individual {
    unsigned char bars[MAX_GENES];                              10
    double fitness;
    int frog_index;
    unsigned char pattern[1024]; // pattern displayed
};

#define MAX_ROULETTE 3000
#define GA_PRESENT_START 362
#define GA_PRESENT_END 618
    // centered at 490 4px width at 64 genes
                                                                20
extern struct individual GA_individuals[POPULATION];
extern struct frog_data GA_frogdata[POPULATION];
extern double *goal;

void init_ga_vars(void);
void usleep(unsigned int msecs );
void ga_present_individual(int i);
void ga_eval_individual(int i);
void ga_eval_population(void);
```

```
void GA_find_time_center(void);
void ga_evolve_population(void);

int gacomp(const void *, const void *);
void ga_evolve_generation(void);

#endif
```

## C.2 Video Frog interface

### C.2.1 videofrog.cpp

```
#include "stdafx.h"
#include "resource.h"
#include "videofrog.h"
#include "ConStream.h"
#include <windows.h>
#include <math.h>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include "main.h"
#include <direct.h>

extern ConStream Log;

MPGetPulseWidth GetPulseWidth;
MPGetIntensityandPhase GetIntensityandPhase;
MPReturnSpec ReturnSpec;
MPReturnPulseandGatewAParams ReturnPulseandGatewAParams;
MPReturnPulsewAParams ReturnPulsewAParams;
MPReturnPulse ReturnPulse;
MPReturnPulseParams ReturnPulseParams;
MPGetSize GetSize;

/* frog globals */
struct frog_data single_frog;
double axisparams[6]; // axis params. time (fs), freq (PetaHertz)
double THz_freqparams[3];
unsigned int frog_trace_size=64;
char gotaxis=0;
double centerfreq_nm=800; // 800nm
double centerfreq_hz=375; // 375 THz
```

```
double nm_to_THz(double nm) {
        double thz;
        thz = 3e8/(800*1e-9)/1e12;
        return thz;
}

double THz_to_nm(double thz) {
        double nm;
        nm = 3e17/(thz*1e12);
        return nm;
}


void Load_VideoFrog_Libraries(void)
{
    HMODULE hMod;

LoadLibrary(
    "C:\\program files\\MesaPhotonics\\VideoFROG64\\libguide40.dll");
LoadLibrary("C:\\program files\\MesaPhotonics\\VideoFROG64\\IPPS20.dll");
hMod = LoadLibrary(
    "C:\\program files\\MesaPhotonics\\VideoFROG64\\PCGPMonitor.dll");

GetPulseWidth = (MPGetPulseWidth) GetProcAddress(hMod,"GetPulseWidth");
    Log << "gpa GetPulseWidth: " << GetPulseWidth << endl;
    GetIntensityandPhase = (MPGetIntensityandPhase)
        GetProcAddress(hMod,"GetIntensityandPhase");
    Log << "gpa GetIntensityandPhase: " << GetIntensityandPhase << endl;
    ReturnSpec = (MPReturnSpec) GetProcAddress(hMod,"ReturnSpec");
    Log << "gpa ReturnSpec: " << ReturnSpec << endl;
    ReturnPulseandGatewAParams = (MPReturnPulseandGatewAParams)
        GetProcAddress(hMod,"ReturnPulseandGatewAParams");
    Log << "gpa ReturnPulseandGatewAParams: " <<
        ReturnPulseandGatewAParams << endl;
    ReturnPulsewAParams = (MPReturnPulsewAParams)
        GetProcAddress(hMod,"ReturnPulsewAParams");
    Log << "gpa ReturnPulsewAParams: " << ReturnPulsewAParams << endl;
    ReturnPulse = (MPReturnPulse) GetProcAddress(hMod,"ReturnPulse");
    Log << "gpa ReturnPulse: " << ReturnPulse << endl;
    ReturnPulseParams = (MPReturnPulseParams)
        GetProcAddress(hMod,"ReturnPulseParams");
    Log << "gpa ReturnPulseParams: " << ReturnPulseParams << endl;
    GetSize = (MPGetSize) GetProcAddress(hMod,"GetSize");
    Log << "gpa GetSize: " << GetSize << endl;
```

```
}

void RetrievePulse(struct frog_data *FD)
{
    unsigned int i;
// double Ttotal, Ftotal;

    FD->PW = GetPulseWidth();
    Log << "pulse width " << FD->PW << endl;
    frog_trace_size = GetSize();
    Log << "trace size " << frog_trace_size << endl;
    if (frog_trace_size) {

    if (!FD->pulse) FD->pulse =
        (double*)malloc(sizeof(double)*frog_trace_size*2);
    if (!FD->gate) FD->gate =
        (double*)malloc(sizeof(double)*frog_trace_size*2);
    if (!FD->tdI) FD->tdI =
        (double*)malloc(sizeof(double)*frog_trace_size);
    if (!FD->tdP) FD->tdP =
        (double*)malloc(sizeof(double)*frog_trace_size);
    if (!FD->fdI) FD->fdI =
        (double*)malloc(sizeof(double)*frog_trace_size);
    if (!FD->fdP) FD->fdP =
        (double*)malloc(sizeof(double)*frog_trace_size);
    if (!FD->freq) FD->freq =
        (double*)malloc(sizeof(double)*frog_trace_size);
    if (!FD->abs_td) FD->abs_td =
        (double*)malloc(sizeof(double)*frog_trace_size);
    if (!FD->abs_fd) FD->abs_fd =
        (double*)malloc(sizeof(double)*frog_trace_size);

    ReturnPulseParams(FD->PulseParams);
    Log << "Pulse Params " << FD->PulseParams[0] << "," <<
        FD->PulseParams[1] << "," << FD->PulseParams[2] <<
        "," << FD->PulseParams[3] << endl;

// pulse width (fs)  bandwidth (nm) autocorrelation (fs)  Time-BW product

    if (!gotaxis) {
        ReturnPulseandGatewAParams(FD->pulse,FD->gate,axisparams);
        for(i=0;i<3;i++)
            THz_freqparams[i] = axisparams[i+3] * 1000;
                // axisparams
        gotaxis = 1;
```

```
    }

    for(i=0;i<frog_trace_size;i++)
        FD->freq[i] =
    THz_to_nm(centerfreq_hz+THz_freqparams[1]+i*THz_freqparams[0]);

    ReturnPulse(FD->pulse,FD->tdI,FD->tdP,FD->fdI,FD->fdP);

                                                                    130
    for(i=0;i<frog_trace_size;i++) {
        FD->abs_td[i] = sqrt(FD->tdI[i]*FD->tdI[i]*
            cos(FD->tdP[i])*cos(FD->tdP[i])+FD->tdI[i]*
            FD->tdI[i]*sin(FD->tdP[i])*sin(FD->tdP[i]));
        FD->abs_fd[i] = sqrt(FD->fdI[i]*FD->fdI[i]*
            cos(FD->fdP[i])*cos(FD->fdP[i])+FD->fdI[i]*
            FD->fdI[i]*sin(FD->fdP[i])*sin(FD->fdP[i]));
    }
    }
    else                                                            140
        Log << "video frog not running" << endl;
}
```

## C.2.2 videofrog.h

```
/* videofrog.h */

#ifndef _VIDEOFROG_H
#define _VIDEOFROG_H

#define WINAPI __stdcall

typedef double (__stdcall *MPGetPulseWidth)(void);
typedef int (__stdcall *MPGetSize)(void);
typedef void (__stdcall *MPReturnPulseParams)(double *);                10
typedef void (__stdcall *MPReturnPulse)(double *, double *,
          double *, double*, double*);
typedef void (__stdcall *MPReturnPulseandGatewAParams)(double *,
          double *, double *);
typedef void (__stdcall *MPReturnPulsewAParams)(double *, double *,
          double *, double*, double*, double *);
typedef void (__stdcall *MPReturnSpec)(double *);
```

```
typedef void (__stdcall *MPGetIntensityandPhase)(double *, double *,
        double *, double*, double*, int);

extern MPGetPulseWidth GetPulseWidth;
extern MPGetIntensityandPhase GetIntensityandPhase;
extern MPReturnSpec ReturnSpec;
extern MPReturnPulseandGatewAParams ReturnPulseandGatewAParams;
extern MPReturnPulsewAParams ReturnPulsewAParams;
extern MPReturnPulse ReturnPulse;
extern MPReturnPulseParams ReturnPulseParams;
extern MPGetSize GetSize;

extern struct frog_data single_frog;
extern double axisparams[6];
   // axis params. time (fs), freq (PetaHertz)
extern double THz_freqparams[3];
extern unsigned int frog_trace_size;
extern char gotaxis;
extern double centerfreq_nm; // 800nm
extern double centerfreq_hz; // 375 THz

struct frog_data {
   double PW;
   double PulseParams[4];
   double *pulse; // complex raw, retrieved pulse
   double *gate;
   double *tdI; // time domain pulse intensity
   double *tdP; // time domain phase
   double *fdI;
      // frequency domain intensity (pulse spectrum)
   double *fdP; // frequency domain phase
   double *freq;  // freq in nm
   double *abs_fd; // absolute value in frequency domain
   double *abs_td; // absolute value in time domain
   double power_meter; // external power meter
};

void Load_VideoFrog_Libraries(void);
double nm_to_THz(double nm);
double THz_to_nm(double thz);
void RetrievePulse(struct frog_data *FD);

#endif
```

## C.3 Main loop

### C.3.1 main.cpp

```
// main.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include "videofrog.h"
#include <io.h>
#include <fcntl.h>
#include "ConStream.h"
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <direct.h>
#include <errno.h>
#include <iostream>
#include <math.h>
#include <time.h>
#include <windows.h>
#include "ga.h"
#include "main.h"

ConStream Log;

/* screen global */
unsigned char *last_loaded_image_ptr=0;
unsigned char last_loaded_bars[1024];
unsigned char *last_saved_image_ptr=0;

void start_GA(void);
void load_goal_from_file(char *fname);
void new_capture_dir_and_chdir();
void stop_GA(void);

// pwd
char pwd[200];
char pwd_current[200];
int GA_run_num = 1;
int GA_ind_num = 0;
int GA_gen_num = 0;
```

```
// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];
TCHAR szWindowClass[MAX_LOADSTRING];
LPPICTURE gpPicture;
HWND ghWnd;
HWND console_hWnd;

// Foward declarations of functions included in this code module:
ATOM            MyRegisterClass(HINSTANCE hInstance);                    50
BOOL            InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR  lpCmdLine,
                     int        nCmdShow)
{

    MSG msg;                                                             60
    init_stuff();

    // open debug window Code in ConStream.cpp
    Log.Open();
    Log << "debug window initialized" << endl;

    Load_VideoFrog_Libraries();

    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);       70
    LoadString(hInstance, IDC_LOADPIC, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    blank_image();                                                       80
    SetWindowPos(console_hWnd,HWND_TOP,600,
            600,400,400,SWP_NOSIZE);

    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0))
```

```
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
                                                                    90
    if (gpPicture)
        gpPicture->Release();

    return (int)msg.wParam;
}

void OutConStr(char *buf)
{
    Log << buf;
}                                                                   100

void dump_individual_data(struct frog_data *fdp, int ind,
    unsigned char *pattern)
{
    FILE *f;
    char fname[200];
    int i;

    sprintf_s(fname,200,"_data\\r%dg%di%d.pp",GA_run_num,
            GA_gen_num,ind); // pp = pulse parameters               110
    if ((fopen_s(&f,fname,"w")==0)) {
        fprintf(f,"%f\t%f\t%f\t%f\n",fdp->PulseParams[0],
            fdp->PulseParams[1],fdp->PulseParams[2],
            fdp->PulseParams[3]);
        fclose(f);
    }

    sprintf_s(fname,200,"_data\\r%dg%di%d.td",GA_run_num,
            GA_gen_num,ind); // td = time domain
// time (fs) <tab> pulse Intensity <tab> pulse Phase <cr>           120
    if ((fopen_s(&f,fname,"w")==0)) {
        for(i=0;i<frog_trace_size;i++)
            fprintf(f,"%f\t%f\t%f\n",
            axisparams[1]+i*axisparams[0],
            fdp->tdI[i],fdp->tdP[i]);
        fclose(f);
    }

    sprintf_s(fname,200,"_data\\r%dg%di%d.fd",GA_run_num,
            GA_gen_num,ind); // fd = freq domain                   130
```

```
// freq (nm) <tab> pulse Intensity <tab> pulse Phase <cr>
    if ((fopen_s(&f,fname,"w")==0)) {
        for(i=0;i<frog_trace_size;i++)
            fprintf(f,"%f\t%f\t%f\n",fdp->freq[i],
                fdp->fdI[i],fdp->fdP[i]);
        fclose(f);
    }

    sprintf_s(fname,200,"_data\\r%dg%di%d.ap",
        GA_run_num,GA_gen_num,ind); // ap = axis parameters
    if ((fopen_s(&f,fname,"w")==0)) {
        fprintf(f,"%f\t%f\t%f\t%f\t%f\t%f\n",
        axisparams[0],axisparams[1],axisparams[2],axisparams[3],
            axisparams[4],axisparams[5]);
        fclose(f);
    }

    sprintf_s(fname,200,"_data\\r%dg%di%d.at",
        GA_run_num,GA_gen_num,ind); // at = absolute time
// time (fs) <tab> absolute time intensity <cr>
    if ((fopen_s(&f,fname,"w")==0)) {
        for(i=0;i<frog_trace_size;i++)
            fprintf(f,"%f\t%f\n",
            axisparams[1]+i*axisparams[0],fdp->abs_td[i]);
        fclose(f);
    }

    sprintf_s(fname,200,"_data\\r%dg%di%d.af",
        GA_run_num,GA_gen_num,ind); // af = absolute freq
    // freq (nm) <tab> absolute frequency intensity <cr>
    if ((fopen_s(&f,fname,"w")==0)) {
        for(i=0;i<frog_trace_size;i++)
            fprintf(f,"%f\t%f\n",fdp->freq[i],fdp->abs_fd[i]);
        fclose(f);
    }

    sprintf_s(fname,200,"_data\\r%dg%di%d.brs",GA_run_num,
            GA_gen_num,ind); // af = absolute freq
            // bar values <cr> 1024 of them
    if ((fopen_s(&f,fname,"w")==0)) {
        for(i=0;i<1024;i++)
            fprintf(f,"%d\n",(int)pattern[i]);
        fclose(f);
    }
```

```
}

void RecordHusimi(struct frog_data *FB, struct frog_data *SB,
        struct frog_data *TB, struct frog_data *W,int num)                    180
{
    FILE *f;
    unsigned int i;
    char lfname[200];

    sprintf_s(lfname,200,"husimi_%d",num);

    if (frog_trace_size) {
    if ((fopen_s(&f,lfname,"w")==0)) {
        // write the husimi file umich format                                 190
        for(i=0;i<frog_trace_size;i++) {
// wavelength (nm), first best freq amp, freq phase,
// second best freq amp, phase, 3b amp, phase, worst amp, phase
            fprintf_s(f,"%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\n",
            FB->freq[i],FB->fdI[i],FB->fdP[i],SB->fdI[i],
            SB->fdP[i],TB->fdI[i],TB->fdP[i],W->fdI[i],W->fdP[i]);
        }
        fclose(f);
    }
    }                                                                         200
}

void capture_pulse()
{
    new_capture_dir_and_chdir();

    RetrievePulse(&single_frog);
    RecordHusimi(&single_frog,&single_frog,&single_frog,&single_frog,1);
    dump_individual_data(&single_frog,0,last_loaded_bars);
}                                                                             210

ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = (WNDPROC)WndProc;
    wcex.cbClsExtra    = 0;                                                   220
```

```
    wcex.cbWndExtra = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon = LoadIcon(hInstance, (LPCTSTR)IDI_LOADPIC);
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH) 0;
    wcex.lpszMenuName = (LPCSTR)IDC_LOADPIC;
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

    return RegisterClassEx(&wcex);                                    230
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;

   hInst = hInstance; // Store instance handle in our global variable
   gpPicture = NULL;

   ghWnd = hWnd = CreateWindow(szWindowClass, szTitle,                240
    WS_OVERLAPPEDWINDOW, 0, 0, 950, 750, NULL,
    NULL, hInstance, NULL);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);                                                250

   return TRUE;
}

void makeImageHeader(unsigned char *ptr) // make a bitmap image header
    // I used bitmaps to avoid windows image libraries.
{
    int i;
    static unsigned char bmpheader[] = {66,77,54,4,12,0,
       0,0,0,0,54,4,0,0,                                               260
              40,0,0,0,0,4,0,0,0,3,0,0,1,0,8,0,0,0,
              0,0,0,0,0,0,196,14,0,0,196,14,0,0,0,
              1,0,0,0,1,0,0,};
              /* 1024x768 8 bit bmp header */
    static unsigned char cmap[1024];
```

```
    static unsigned char cmap_initialized = 0;

    if (!cmap_initialized) {
        for(i=0;i<256;i++) {
            cmap[i*4] = i;                                          270
            cmap[i*4+1] = i;
            cmap[i*4+2] = i;
            cmap[i*4+3] = 0;
        }
        cmap_initialized = 1;
    }

    memcpy(ptr,bmpheader,54);
    ptr += 54;
    memcpy(ptr,cmap,1024);                                          280
}


char *makeBarsImage(unsigned char *ptr, int pxStart, int pxEnd,
        unsigned char *pattern, int pattern_len)
{
    static char barvalues[1024];
    int barwidth;
    int i,j;
                                                                    290
    barwidth = (pxEnd-pxStart)/pattern_len;
    makeImageHeader(ptr);
    ptr += 1024+54;

    for(i=0;i<1024;i++) {
        if (i<pxStart || i>pxEnd)
            barvalues[i] = 0;
        else
            barvalues[i] = pattern[(i-pxStart)/barwidth];
    }                                                               300

    for(j=0;j<768;j++) {
        for(i=0;i<1024;i++) {
            *ptr = barvalues[i];
            ptr++;
        }
    }
    memcpy(last_loaded_bars,barvalues,1024);
    return(barvalues);
}                                                                   310
```

```
void makeRandBars(unsigned char *ptr)
{
    unsigned char gastr[60];
    int i;
    char *pp;

    for(i=0;i<60;i++)
        gastr[i] = (rand()*255)/RAND_MAX;
    pp = makeBarsImage(ptr,0,1024,gastr,60);                    320
}


// This function loads a file into an IStream.
void LoadImageData(unsigned char *ptr)
{
    // save last load
    last_loaded_image_ptr = ptr;

    // get file size                                            330
    DWORD dwFileSize = BMP_IMG_SIZE;

    LPVOID pvData = NULL;
    // alloc memory based on file size
    HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwFileSize);
    _ASSERTE(NULL != hGlobal);

    pvData = GlobalLock(hGlobal);
    _ASSERTE(NULL != pvData);
                                                             340
    memcpy(pvData,ptr,BMP_IMG_SIZE);

    GlobalUnlock(hGlobal);

    LPSTREAM pstm = NULL;
    // create IStream* from global memory
    HRESULT hr = CreateStreamOnHGlobal(hGlobal, TRUE, &pstm);
    _ASSERTE(SUCCEEDED(hr) && pstm);

    // Create IPicture from image file                          350
    if (gpPicture)
        gpPicture->Release();
    hr = ::OleLoadPicture(pstm, dwFileSize, FALSE,
        IID_IPicture, (LPVOID *)&gpPicture);
    _ASSERTE(SUCCEEDED(hr) && gpPicture);
```

```
    pstm->Release();

    InvalidateRect(ghWnd, NULL, TRUE);
}
                                                                    360
VOID CALLBACK scan_step(
    HWND hwnd,   // handle to window for timer messages
    UINT message,  // WM_TIMER message
    UINT idTimer,  // timer identifier
    DWORD dwTime) // current system time
{
    static unsigned char *ptr=0;
    char *pp;

    if (!ptr)                                                       370
        {
        ptr = (unsigned char*)malloc(BMP_IMG_SIZE);
        makeRandBars(ptr);
        }

    switch(idTimer) {
        case IDT_TIMER_RANDBARS:
        stop_rand_bars();

        makeRandBars(ptr);                                          380

        start_rand_bars();
        break;

        case IDT_TIMER_SCANLR:
            {
            static int spos=0;
            static unsigned char bars[128];
            double d;
            FILE *f;                                                390

            stop_scan(IDT_TIMER_SCANLR);

            if ((fopen_s(&f,"scanlr.txt","a+")==0))
            {
                d = (GetPulseWidth)();
                Log << spos << " " << d << endl;
                fprintf(f,"%d %f\n",spos,d);
                fclose(f);
            }                                                       400
```

```
            else
            {
                Log << "Can't open scanlr.txt" << endl;
            }

            memset(bars,0,128);
            if (spos>127) spos = 0;
            bars[spos] = 255;
            pp = makeBarsImage(ptr,0,1024,bars,128);
            spos++;

            start_scan(IDT_TIMER_SCANLR);
            }
        break;
    }

    LoadImageData(ptr);
}

void new_GA(void)
{
    Log << "starting new GA run" << endl;
    new_garun_dir_and_chdir();
    GA_ind_num = 0;
    GA_gen_num = 0;
    init_ga_vars();
}

VOID CALLBACK ga_step(
    HWND hwnd,   // handle to window for timer messages
    UINT message,  // WM_TIMER message
    UINT idTimer,  // timer identifier
    DWORD dwTime) // current system time
{
    stop_GA();

    if (GA_ind_num<=POPULATION) {

        if (GA_ind_num==0 && GA_gen_num==0) {
            GA_find_time_center();
        }

        if (GA_ind_num==0) {
        ga_present_individual(GA_ind_num);
        }
```

```
        else {
        ga_eval_individual(GA_ind_num-1);
        if (GA_ind_num<POPULATION) ga_present_individual(GA_ind_num);
        }
        GA_ind_num++;                                                    450
    }
    else {
        int i;
        ga_eval_population();
        RecordHusimi(&(GA_frogdata[GA_individuals[0].frog_index]),
            &(GA_frogdata[GA_individuals[1].frog_index]),
            &(GA_frogdata[GA_individuals[2].frog_index]),
        &(GA_frogdata[GA_individuals[POPULATION-1].frog_index]),
            GA_gen_num);
        for(i=0;i<10;i++)                                                460
    dump_individual_data(&(GA_frogdata[GA_individuals[i].frog_index]),
        i,GA_individuals[i].pattern);
        ga_evolve_population();
        GA_ind_num = 0;
        GA_gen_num++;
        Log << "generation " << GA_gen_num << endl;
    }

    start_GA();
}                                                                        470

void init_stuff()
{
    srand( (unsigned)time( NULL ) );
    memset(&single_frog,0,sizeof(struct frog_data));
    pwd[0] = 0;
    init_ga_vars();
}

void start_scan(UINT_PTR timernum)                                       480
{
    SetTimer(ghWnd,timernum,1000,(TIMERPROC)scan_step);
}

void stop_scan(UINT_PTR timernum)
{
    KillTimer(ghWnd,timernum);
}

void start_GA(void)                                                      490
```

```
{
    SetTimer(ghWnd,IDT_TIMER_GASTEP,4000,(TIMERPROC)ga_step);
}

void stop_GA(void)
{
    KillTimer(ghWnd,IDT_TIMER_GASTEP);
}

                                                                    500
void stop_timers()
{
    KillTimer(ghWnd,IDT_TIMER_SCANLR);
    KillTimer(ghWnd,IDT_TIMER_RANDBARS);
    KillTimer(ghWnd,IDT_TIMER_GASTEP);
}

void start_rand_bars()
{
// ReRegisterClass(hInst);                                          510
    SetTimer(ghWnd,IDT_TIMER_RANDBARS,500,(TIMERPROC)scan_step);
}

void stop_rand_bars()
{
    KillTimer(ghWnd,IDT_TIMER_RANDBARS);
}

void write_bmp_to_file(unsigned char *ptr)
{                                                                   520
    FILE *f;
    static int bmpnum=0;
    char fname[200];

    sprintf_s(fname,200,"saved_%d.bmp",bmpnum);
    if ((fopen_s(&f,fname,"w")==0)) {
        fwrite(ptr,BMP_IMG_SIZE,1,f);
        fclose(f);
        bmpnum++;
    }                                                               530
}

void load_bmp_from_file(char *fname, unsigned char *ptr)
{
    FILE *f;
```

```
    if ((fopen_s(&f,fname,"r")==0)) {
        fread(ptr,BMP_IMG_SIZE,1,f);
        fclose(f);
    }
}

void load_goal_from_file(char *fname)
{
    FILE *f;
    char buf[512];
    char *ptr;
    int i;

    if ((fopen_s(&f,fname,"r")==0)) {
        Log << "opened goal file " << fname << endl;
        for(i=0;i<frog_trace_size;i++)
            if (fgets(buf,512,f)) {
            buf[strlen(buf)-1] = 0;
            ptr = strchr(buf,' ');
            ptr++;
            goal[i] = atof(ptr);
            Log << buf << " read " << goal[i] << endl;
            }
    fclose(f);
    }
}


void PaintWindow(HWND hWnd)
{
    PAINTSTRUCT ps;
    HDC hdc;

    hdc = BeginPaint(hWnd, &ps);
    if (gpPicture)
    {
        // get width and height of picture
        long hmWidth;
        long hmHeight;
        gpPicture->get_Width(&hmWidth);
        gpPicture->get_Height(&hmHeight);
        // convert himetric to pixels
    int nWidth = MulDiv(hmWidth, GetDeviceCaps(hdc, LOGPIXELSX),
            HIMETRIC_INCH);
```

```
    int nHeight = MulDiv(hmHeight, GetDeviceCaps(hdc, LOGPIXELSY),
        HIMETRIC_INCH);
    RECT rc;
    GetClientRect(hWnd, &rc);
    // display picture using IPicture::Render
    gpPicture->Render(hdc, 0, 0, nWidth, nHeight, 0,
        hmHeight, hmWidth, -hmHeight, &rc);
  }
  EndPaint(hWnd, &ps);
}                                                                    590

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
        LPARAM lParam)
{
  int wmId, wmEvent;

  switch (message)
  {
  case WM_COMMAND:
    wmId  = LOWORD(wParam);                                          600
    wmEvent = HIWORD(wParam);
    // Parse the menu selections:
    switch (wmId)
    {
    case ID_SCREEN_LOADIMGFROM:
      {
      // get file name to open
      TCHAR szFile[MAX_PATH];
      ZeroMemory(szFile, MAX_PATH);
      OPENFILENAME ofn;                                              610
      ZeroMemory(&ofn, sizeof(OPENFILENAME));
      ofn.lStructSize = sizeof(OPENFILENAME);
      ofn.Flags = OFN_FILEMUSTEXIST |
        OFN_PATHMUSTEXIST | OFN_HIDEREADONLY;
      ofn.hwndOwner = hWnd;
      ofn.lpstrFilter =
_T("Supported Files Types(*.bmp)\0*.bmp;\0Bitmaps (*.bmp)\0*.bmp\0\0");
      ofn.lpstrTitle  = _T("Open bmp File");
      ofn.lpstrFile   = szFile;
      ofn.nMaxFile  = MAX_PATH;                                      620
      if (IDOK == GetOpenFileName(&ofn)) {
      if (!last_saved_image_ptr)
        last_saved_image_ptr =
          (unsigned char*)malloc(BMP_IMG_SIZE);
        load_bmp_from_file(szFile,
```

```
                last_saved_image_ptr);
            LoadImageData(last_saved_image_ptr);
            }
        _chdir(pwd_current);
        }
    break;

    case ID_LOADGOAL:
        {
        // get file name to open
        TCHAR szFile[MAX_PATH];
        ZeroMemory(szFile, MAX_PATH);
        OPENFILENAME ofn;
        ZeroMemory(&ofn, sizeof(OPENFILENAME));
        ofn.lStructSize = sizeof(OPENFILENAME);
        ofn.Flags = OFN_FILEMUSTEXIST |
            OFN_PATHMUSTEXIST | OFN_HIDEREADONLY;
        ofn.hwndOwner = hWnd;
        ofn.lpstrFilter =
_T("Supported Files Types(*.*)\0*.*;\0Goal (*.*)\0*.*\0\0");
        ofn.lpstrTitle  = _T("Open bars File");
        ofn.lpstrFile   = szFile;
        ofn.nMaxFile    = MAX_PATH;
        if (IDOK == GetOpenFileName(&ofn)) {
            if (!goal)
    goal = (double*)malloc(sizeof(double)*frog_trace_size);
            load_goal_from_file(szFile);
            }
        _chdir(pwd_current);
        }
    break;

    case IDM_RAND_GO:
        stop_timers();
        start_rand_bars();
        break;
    case IDM_RAND_STOP:
        stop_rand_bars();
        break;
    case IDM_SCAN_GO:
        stop_timers();
        start_scan(IDT_TIMER_SCANLR);
        break;
    case IDM_SCAN_STOP:
        stop_scan(IDT_TIMER_SCANLR);
```

```
        break;
    case IDM_SAVEPULSE:
        save_pulse();
        break;
    case ID_CAPTURE_RECORDPULSE:
        capture_pulse();
        break;
    case ID_FILE_NEWDATADIRECTORY:
        new_data_dir_and_chdir();
        break;                                                          680
    case ID_GENETICALGORITHM_START:
        start_GA();
        break;
    case ID_GENETICALGORITHM_STOP:
        stop_GA();
        break;
    case ID_GENETICALGORITHM_NEWRUN:
        new_GA();
        break;
    case ID_SCREEN_LOAD:                                                690
if (last_saved_image_ptr) LoadImageData(last_saved_image_ptr);
        break;
    case ID_SCREEN_SAVE:
        if (last_loaded_image_ptr) {
if (!last_saved_image_ptr)
last_saved_image_ptr = (unsigned char*)malloc(BMP_IMG_SIZE);
memcpy(last_saved_image_ptr,last_loaded_image_ptr,BMP_IMG_SIZE);
        }
    break;
    case ID_SCREEN_CLEAR:                                               700
        blank_image();
    break;
    case IDM_EXIT:
        DestroyWindow(hWnd);
    break;
    default:
      return DefWindowProc(hWnd, message, wParam, lParam);
    break;
case WM_PAINT:
    PaintWindow(hWnd);                                                  710
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
```

```
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
                                                                    720

void blank_image()
{
    static unsigned char *ptr=0;
    unsigned char bars[128];

    if (!ptr) {
        ptr = (unsigned char*)malloc(BMP_IMG_SIZE);
        memset(bars,0,128);
        makeBarsImage(ptr,0,1024,bars,128);                         730
    }
    LoadImageData(ptr);
}

void new_garun_dir_and_chdir()
{
    char dpath[200];

    if (!pwd[0]) new_data_dir_and_chdir();
    sprintf_s(dpath,200,"%s\\run_%d",pwd,GA_run_num);               740
    _mkdir(dpath);
    _chdir(dpath);
    memccpy(pwd_current,dpath,0,200);
    sprintf_s(dpath,200,"%s\\run_%d\\_data",pwd,GA_run_num);
    _mkdir(dpath);
    GA_run_num++;

    Log << "path: " << dpath << endl;
}
                                                                    750
void new_capture_dir_and_chdir()
{
    char dpath[200];
    static int capture=0;

    if (!pwd[0]) new_data_dir_and_chdir();
    sprintf_s(dpath,200,"%s\\capture_%d",pwd,capture);
    _mkdir(dpath);
    _chdir(dpath);
    memccpy(pwd_current,dpath,0,200);                               760
```

```
    sprintf_s(dpath,200,"%s\\capture_%d\\_data",pwd,capture);
    _mkdir(dpath);
    capture++;

    Log << "path: " << dpath << endl;
}



void new_data_dir_and_chdir()                                          770
{
    char dpath[200];
    time_t now;
    struct tm tme;

    time(&now);
    localtime_s(&tme,&now);

    sprintf_s(dpath,200,"%s%02d%02d%02d%02d%02d",DATA_DIR,
    tme.tm_year-100,tme.tm_mon+1,tme.tm_mday,tme.tm_hour,tme.tm_min);  780
    _mkdir(dpath);
    _chdir(dpath);
    memcpy(pwd,dpath,200);

    Log << "path: " << dpath << endl;
}
```

## C.3.2 main.h

```
#ifndef _MAIN_H
#define _MAIN_H

#include <windows.h>

#define _WIN32_WINNT 0x0501
#define WINVER 0x0501
#define NTDDI_VERSION NTDDI_WINXP

#define DATA_DIR "c:\\data\\2006\\"                                    10

#define MAX_LOADSTRING 100
#define HIMETRIC_INCH 2540
```

```
#define MAP_LOGHIM_TO_PIX(x,ppli)
   ( ((ppli)*(x) + HIMETRIC_INCH/2) / HIMETRIC_INCH )
#define BMP_IMG_SIZE 787510

#define IDT_TIMER_RANDBARS 1
#define IDT_TIMER_SCANLR 2
#define IDT_TIMER_SCANTB 3
#define IDT_TIMER_GASTEP 4

extern HWND ghWnd;
extern int GA_gen_num;

void blank_image();
void init_stuff();
int read_field_max_power_meter(double *power);
void PaintWindow(HWND hWnd);
void start_scan(UINT_PTR);
void stop_scan(UINT_PTR);
void start_rand_bars();
void stop_rand_bars();
void LoadImageData(unsigned char *ptr);
char *makeBarsImage(unsigned char *ptr, int pxStart,
   int pxEnd, unsigned char *pattern, int pattern_len);
void new_data_dir_and_chdir();
void new_garun_dir_and_chdir();
void stop_timers();
void OutConStr(char*);

#endif
```

# Appendix D

# Code to Emulate a LP 120 Lamp

lamp.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

/* change this definition for the correct port */
#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

main()
{
  int fd,c, res;
  struct termios oldtio,newtio;
  char buf[255];
  FILE *f;
  char start=0;
  int cnt = 0;
  char ss[23];
  char reseted = 0;

  ss[0] = 127;
  ss[1] = 183;
```

```
 ss[2] = 21;
 ss[3] = 182;                                                    30

fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
if (fd <0) {perror(MODEMDEVICE); exit(-1); }
tcflush(fd, TCIFLUSH);

f = fdopen(fd, "w+");
while (STOP==FALSE) {
      /* loop until we have a terminating condition */
   res = fgetc(f);
   if (res>0)                                                    40
      {
      printf("%d - %d\n",cnt,res);
            {
            if (cnt<4)
                  fprintf(f,"%c",ss[cnt]);
            else
                  fprintf(f,"%c",182);
            fflush(f);
            cnt++;
            if (reseted<40)                                      50
                  {
                  reseted++;
                  printf("resetting connection\n");
                  close(fd);
                  usleep(100000);
                  fd = open(MODEMDEVICE, O_RDWR|O_NOCTTY|
                        O_NONBLOCK);
                  tcflush(fd, TCIFLUSH);
                  f = fdopen(fd, "w+");
                  }                                              60
            }
      }
 }
 /* restore the old port settings */
}
```

# Bibliography


[1] T. Brixner, N.H. Damrauer, P. Niklaus, and G. Gerber. Photoselective adaptive femtosecond quantum control in the liquid phase. *Nature*, 414:57–60, 2001.

[2] Herschel Rabitz, Regina de Vivie-Riedle, Marcus Motzkus, and Karl Kompa. Whither the future of controlling quantum phenomea? *Science*, 288:824–8, 2000.

[3] Valentyn I. Prokhorenko, Andrea M. Nagy, Stephen A. Waschuk, Leonid S. Brown, Robert R. Birge, and R. J. Dwayne Miller. Coherent Control of Retinal Isomerization in Bacteriorhodopsin. *Science*, 313:1257, 2006.

[4] Lauri Pekkarinen and Henry Linschitz. Studies on metastable states of porphyrins. ii. spectra and decay kinetics of tetraphenylporphine, zinc tetraphenylporphine and bacteriochlorophyll. *Journal of the American Chemical Society*, 82:2407–2411, 1960.

[5] Waseem Bakr. Optical nuclear polarization for nmr signal enhancement. Bachelor's Thesis, MIT, June 2005.

[6] Herman A. Haus. Mode-locking of lasers. *IEEE Journal on Selected Topics in Quantum Electronics*, 6(6):1173, 2000.

[7] Neil Gershenfeld. *The Nature of Mathematical Modeling.* Cambridge University Press, Cambridge, 1999.

[8] N. Gershenfeld. System employing dissipative pseudorandum dynamics for communications and measurement. U.S. Patent No. 5,729,388.

[9] F.X. Kartner, Juerg Aus der Au, and U. Keller. Mode-locking with slow and fast saturable absorbers–what's the difference? *IEEE Journal on Selected Topics in Quantum Electronics*, 4(2):159–168, March 1998.

[10] F.X. Kartner, I.D. Jung, and U. Keller. Soliton mode-locking with saturable absorbers. *IEEE Journal on Selected Topics in Quantum Electronics*, 2(3):540–556, September 1996.

[11] D.E. Spence, P.N. Kean, and W. Sibbett. 60-fsec pulse generation from a self-mode-locked ti:sapphire laser. *Optics Letters*, 16:42–44, 1991.

[12] Ahmed H. Zewail. Femtochemistry: Atomic-scale dynamics of the chemical bond. *J. Phys. Chem. A*, 104:5660–5694, 2000.

[13] Jason Taylor. Shaped ultrafast optical pumping for nmr applications. Master's thesis, MIT, 2002.

[14] W.S. Warren. The usefulness of nmr quantum computing. *Science*, 277:1688, 1997.

[15] L. Shulman and U. Vazirani. Scalable nmr quantum computation. http://xxx.lanl.gov/abs/quant-ph/9804060, 1998.

[16] Robert A. Reed, Roberto Purrello, Kristine Prendergast, and Thomas G. Spiro. Resonance raman characterization of the radical anion and triplet states of zinc tetraphenylporphine. *J. Phys. Chem.*, 95:9720–9727, 1991.

[17] Herman A. Haus. Theory of mode locking with a fast saturable absorber. *J. Appl. Phys.*, 46:3049–3058, 1975.

[18] A.M. Weiner. Femtosecond pulseshaping using spatial light modulators. *Review of Scientific Instruments*, 71(5):1929–1960, 2000.

[19] N.A. Gershenfeld and I.L. Chuang. Bulk spin-resonance quantum computation. *Science*, 275:350–6, 1997.

[20] M. Iinuma, Y. Takahashi, I. Shake, M. Oda, A. Masaike, and T. Yabuzaki. High proton polarization by microwave-induced optical nuclear polarziation at 77k. *Physical Review Letters*, 84(1):171–174, 2000.

[21] Kazuyuki Takeda, K. Takegoshi, and Takehiko Terao. Dynamic nuclear polarization by electron spins in the photoexcited triplet state: I. attainment of proton polarization of 0.7 at 105 k in napthalene. *Journal of the Physical Society of Japan*, 73(8):2313–2318, 2004.

[22] Thad G. Walker and William Happer. Spin-exchange optical pumping of noble-gas nuclei. *Reviews of Modern Physics*, 69(2):629–642, 1997.

[23] Anthony Harriman. Luminescence of porphyrins and metalloporphyrins. *J.C.S. Faraday I*, 76:1978–1985, 1980.

[24] Kazuyuki Ishii, Satoko Abiko, and Nago Kobayashi. Time-resolved electron spin resonance of gallium and germanium porphyrins in the excited triplet state. *Inorganic Chemistry*, 39:468–472, 2000.

[25] J. Bargon and K.G. Seifert. Nuclear spin polarization from triplet states in solution. *Ber. Bunsen. Phys. Chem.*, 78:1180–1182, 1974.

[26] Dr. Kenneth Brown and Waseem Bakr. personal communcation, 2002.

[27] B. J. Pearson, J. L. White, T. C. Weinacht, and P. H. Bucksbaum. Coherent control using adaptive learning algorithms. *Phys. Rev. A*, 63:063412, 2001.

[28] M.A. Dugan, J.X. Tull, and W.S. Warren. High-resolution acousto-optic shaping of unamplified and amplified femtosecond laser pulses. *J. Opt. Soc. Am. B*, 14(9):2348, 1997.

[29] A. Brodeur and S.L. Chin. Ultrafast white-light continuum generation and self-focusing in transparent condensed media. *J. Opt. Soc. Am. B*, 16(4):637, 1999.

[30] Stephanie Forrest. Genetic algorithms: Principles of natural selection applied to computation. *Science*, 261:872–78, 1993.

[31] J. Paye. The chronocyclic representation of ultrashort light pulses. *IEEE J. Quant. Elect.*, 28:2262–2273, 1992.

[32] G. Cerullo, C.J. Bardeen, Q. Wang, and C.V. Shank. High-power femtosecond chirped pulse excitation of molecules in solution. *Chemical Physics Letters*, 262:362–368, 1996.

[33] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall PTR, 1997.

[34] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical One-Way Functions. *Science*, 297(5589):2026–2030, 2002.

[35] Darwin R. Reyes, Moustafa M. Ghanem, George M. Whitesides, and Andreas Manz. Glow discharge in microfluidic chips for visible analog computing. *Lab on a Chip*, 2:113–116, 2002.

[36] B. Vigoda, J. Dauwels, N. Gershenfeld, and H. Loeliger. Low-complexity lfsr synchronization by forward-only message passing. Submitted to the IEEE Trans. on Information Theory, June 2003.

[37] Benjamin Vigoda. *Analog Logic: Continuous-Time Analog Circuits for Statistical Signal Processing.* PhD thesis, MIT, 2003.

[38] Seth Lloyd. Computational capacity of the universe. *Physical Review Letters*, 88(23):237901, 2002.

[39] Roger W. Brockett. Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems. In *Proc. 27th IEEE Conf. Dec. and Control*, pages 799–803, Austin, TX, Dec. 1988.

[40] C. Moore. Unpredictability and undecidability in dynamical systems. *Phys. Rev. Lett.*, 64(20):2354–2357, 1990.

[41] Jack Bonnell Dennis. *Mathematical Programming and Electrical Networks.* PhD thesis, MIT, 1958.

[42] David B Kirk, Douglas Kerns, Kurt Fleischer, and Alan H. Barr. Analog vlsi implementation of multi-dimensional gradient descent. In *Advances in Neural Information Processing Systems*, volume 5, pages 789–796, San Mateo, CA, 1993. Morgan Kaufman.

[43] A.E. Siegman and D. J. Kuizenga. Modulator frequency detuning effects in the fm mode-locked laser. *IEEE J. Quant. Elect.*, 6:803, 1970.

[44] Anthony E. Siegman. *Lasers.* University Science Books, Sausalito, CA, 1986.

[45] E. P. Ippen. Principles of passive mode locking. *Appl. Phys. B*, 58:159–170, 1994.

[46] Joachim Herrmann. Theory of kerr-lens mode locking: role of self-focusing and radically varying gain. *J. Opt. Soc. Am. B*, 11(3):498, 1994.

[47] R.L. Fork, O. E. Martinez, and J. P. Gordon. Negative dispersion using pairs of prisms. *Opt. Lett.*, 9:150, 1984.

[48] Edmond B. Treacy. Optical pulse compression with diffraction gratings. *IEEE Journal of Quantum Electronics*, 5(9):454, 1969.

[49] Govind P. Agrawal. *Nonlinear Fiber Optics.* Academic Press, third edition, 2001.

[50] Benjamin Vigoda. A nonlinear dynamic system for spread spectrum code acquisition. Master's thesis, MIT, 1999.

[51] Whitaker, Avrampolous, and Huang. Optical linear feedback shift register. U.S. Patent No. 5,208,705.

[52] G. Grinstein and N. Gershenfeld. System employing dissipative pseudorandom dynamics and selective feedback for communications and measurement. U.S. Patent No. 5,579,337.

[53] M.C. Farries and D.N. Payne. Optical fiber switch employing a sagnac interferometer. *Appl. Phys. Lett.*, 55(1):25–26, July 1989.

[54] Edward J. Grace. *Optimising Kerr-lens mode-locked lasers.* PhD thesis, University of London, 2002.

[55] S.A. Hamilton, B.S. Robinson, T.E. Murphy, S.J. Savage, and E.P. Ippen. 100 gb/s optical time-division multiplexed networks. *Journal of Lightwave Technology*, 20(12):2086, 2002.

[56] O. E. Martinez, R.L. Fork, and J. P. Gordon. Theory of passively mode-locked lasers including self-phase modulation and group-velocity disperson. *Opt. Lett.*, 9:156–158, 1984.

[57] J.M. Soto-Crespo, N. Akhmediev, and G. Town. Continuous-wave versus pulse regime in a passively mode-locked laser with a fast saturable absorber. *J. Opt. Soc. Am. B*, 19(2):234–242, 2002.

[58] J.M. Soto-Crespo, N. Akhmediev, and V. Afanasjev. Stability of the pulselike solutions of the quintic complex ginzburg-landau equation. *J. Opt. Soc. Am. B*, 13(7):1439–1449, 1996.

[59] R. Montagne, E. Hernandez-Garcia, and M. San Miguel. Numerical study of a lyapunov functional for the complex ginzburg-landau equation. http://xxx.lanl.gov/abs/cond-mat/9508115, 1995.

[60] Igor S. Aranson and Lorenz Kramer. The world of the complex ginzburg-landau equation. *Reviews of Modern Physics*, 74:99–143, 2002.

[61] E. Lioudakis, K. Adamou, and A. Othonos. Prism-based ultrafast pulse-shaping apparatus. *Optical Engineering*, 44(3):034203–1, March 2005.

[62] C.P. Singh and Sukhdev Roy. Dynamics of all-optical switching in c60 and its application to optical logic gates. *Optical Engineering*, 43(2):426, 2004.

[63] C.P. Singh, K.S Bindra, B. Jain, and S.M. Oak. All-optical switching characteristics of metalloporphyrins. *Optics Communications*, 245:407–414, 2005.

[64] T. Benincori, G. Bongiovanni, C. Botta, G. Cerullo, , G. Lanzani, A. Mura, L. Rossi, F. Sannicolo, and R. Tubino. Tuning of teh excited-state lifetime by control of the structural relaxation in oligothiophenes. *Physical Review B*, 58(14):9082, 1998.

[65] B. Kraabel, D. Moses, and A. J. Heeger. Direct observation of the intersystem crossing in poly(3-octylthiophene). *Journal of Chemical Physics*, 103:5102, 1995.

[66] Martin broglia, Maria Gomez, Sonia Berntolotti, Hernan Montejano, and Carlos Previtali. Photophysical properties of safranine and phenosafranine: A comparitive study by laser flash photolysis and laser induced optoacuoustic spectroscopy. *Journal of Photochemistry and Photobiology A: Chemsitry*, 173:115–120, 2005.

[67] J. David Musgraves, Brett T. Close, and David M. Tanenbaum. A maskless photolithographic prototyping system using a low-cost consumer projector and a microscope. *American Journal of Physics*, 73(10):980, 2005.

[68] Rick Trebino. *Frequency-Resolved Optical Gating: The Measurement of Ultrashort Laser Pulses.* Kluwer Academic Publishers, Boston, MA, 2002.

[69] Oliver Hofmann, Xuhua Wang, Alastair Cornwell, Stephen Beecher, Amal Raja, and Donal D.C. Bradley. Monolithically integrated dye-doped pdms long-pass filters for disposable on-chip fluorescence detection. *Lab Chip*, 6:981–987, 2006.

[70] Yoshihiko Kanemitsu, Naoya Shimizu, Katsunori Suzuki, Yotaro Shiraishi, and Masami Kuroda. Optical and structural properties of oligothiophene crystalline films. *Physical Review B*, 54:2198–2204, 1996.